# 网络基本功系列

wizardforcel

Published with GitBook



VISIT…

LANZAROTE Caliente.COM

# 目錄


| | |
|---|---|
| 介绍 | 0 |
| 网络基本功（一）：细说网络传输 | 1 |
| 网络基本功（二）：细说交换机 | 2 |
| 网络基本功（三）：细说VLAN与Trunk | 3 |
| 网络基本功（四）：细说路由（上） | 4 |
| 网络基本功（五）：细说路由（下） | 5 |
| 网络基本功（六）：链路聚合 | 6 |
| 网络基本功（七）：细说IP地址与子网 | 7 |
| 网络基本功（八）：细说TCP滑动窗口 | 8 |
| 网络基本功（九）：细说TCP重传 | 9 |
| 网络基本功（十）：细说TCP确认机制 | 10 |
| 网络基本功（十一）：TCP窗口调整与流控 | 11 |
| 网络基本功（十二）：细说Linux网络配置（上） | 12 |
| 网络基本功（十三）：细说Linux网络配置（下） | 13 |
| 网络基本功（十四）：细说诊断工具ping | 14 |
| 网络基本功（十五）：细说网络性能监测与实例（上） | 15 |
| 网络基本功（十六）：细说网络性能监测与实例（下） | 16 |
| 网络基本功（十七）：细说tcpdump的妙用（上） | 17 |
| 网络基本功（十八）：细说tcpdump的妙用（下） | 18 |
| 网络基本功（十九）：细说NAT原理与配置 | 19 |
| 网络基本功（二十）：细说ICMP和ARP | 20 |
| 网络基本功（二十一）：细说HTTP（上） | 21 |
| 网络基本功（二十二）：细说HTTP（下） | 22 |
| 网络基本功（二十三）：Wireshark抓包实例诊断TCP连接问题 | 23 |
| 网络基本功（二十四）：Wireshark抓包实例分析TCP重传 | 24 |
| 网络基本功（二十五）：Wireshark抓包实例分析TCP重复ACK与乱序 | 25 |
| 网络基本功（二十六）：Wireshark抓包实例分析TCP窗口及reset | 26 |
| 网络基本功（二十七）：Wireshark抓包实例分析HTTP问题(上) | 27 |
| 网络基本功（二十八）：Wireshark抓包实例分析HTTP问题(下) | 28 |
| 网络基本功（二十九）：Wireshark抓包实例诊断数据库常见问题 | 29 |

网络基本功（三十）：细说DNS（上） 30
网络基本功（三十一）：细说DHCP 31

# 网络基本功系列

> 作者：Zhang Jiawen
>
> 来源：网络基本功系列：细说网络那些事儿

# 网络基本功（一）：细说网络传输

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

常言道：欲练神功，必先练好基本功。之前做了一个关于IP路由，默认网关和掩码的问答贴，做完这个帖子觉得如果对网络知识点做一个系统的阐述，应该会很有帮助。

本系列文章着重于讲解网络管理实际应用中常常涉及的重要知识点，尽量以实用为主。准备写的几个章节暂时有（可能会有增减）：


- 网络传输
- 交换机
- VLAN与Trunk
- 路由(上)
- 路由(下)
- 链路聚合
- IP地址与子网
- NAT原理与配置
- ICMP与ARP
- TCP滑动窗口
- TCP重传
- TCP确认机制
- TCP窗口调整与流控
- Wireshark抓包实例诊断TCP连接问题 (精)
- Wireshark抓包实例诊断TCP重传 (精)
- Wireshark抓包实例诊断TCP重复ACK与乱序 （精）
- Wireshark抓包实例诊断TCP窗口与reset （精）
- Wireshark抓包实例诊断HTTP问题 （精）
- Wireshark抓包实例诊断数据库问题
- HTTP(上)
- HTTP(下)
- DNS（上） （NEW)
- 细说Linux网络配置（上）
- 细说Linux网络配置（下）
- 常用诊断工具：ping

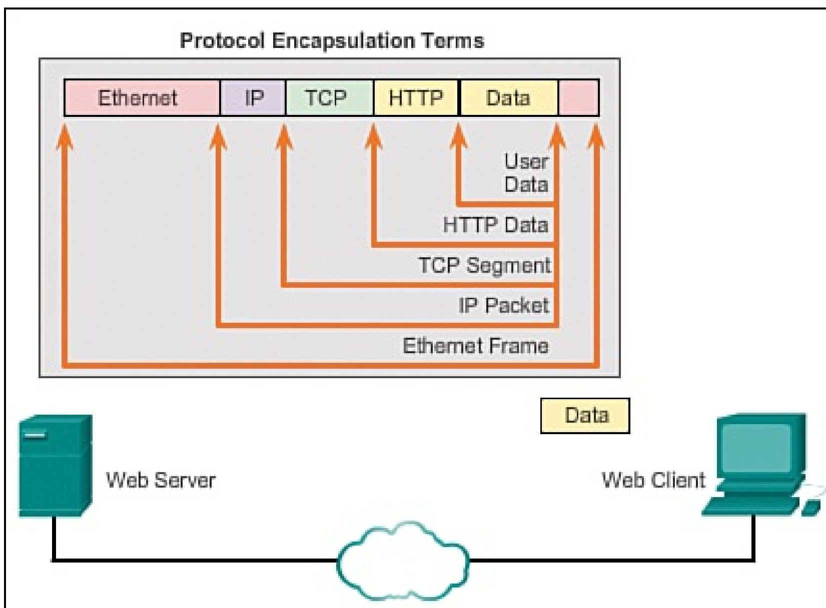
- 网络性能监测与实例（上）
- 网络性能监测与实例（下）
- tcpdump的妙用 (上)
- tcpdump的妙用（下）

# 更多信息

首先来看一个例子：

示例：网络服务器向客户端传送数据的过程:

在详细阐述网络传输过程之前，先来看一个最常见的例子，下图显示了一个网络服务器向客户端传送数据的完整过程：




```
1\. 需要传送的数据是网络服务器的HTML页面。

2\. 应用协议HTTP报文头添加到HTML数据之前。报文头信息包括：服务器所使用的HTTP版本，以及表明它包含发给网

3\. HTTP应用层协议将HTML格式的网页数据发送给传输层。TCP传输层用于管理网络服务器和客户端之间的会话。

4\. IP信息添加到TCP信息之前。IP指定适当的源和目的IP地址。这些信息就构成了IP报文。

5\. 以太网协议添加到IP报文的两端之后，就形成了数据链路帧。上述帧发送至通向网络客户端的路径上的最近一个路

6\. 数据通过互联网络传输，互联网络包含媒介和中间设备。

7\. 客户端接收到包含数据的数据链路帧，处理各层协议头，之后以添加时相反的顺序移除协议头。首先处理并移除以

8\. 之后，将网页信息传递给客户端网页浏览器软件。
```

数据封装:

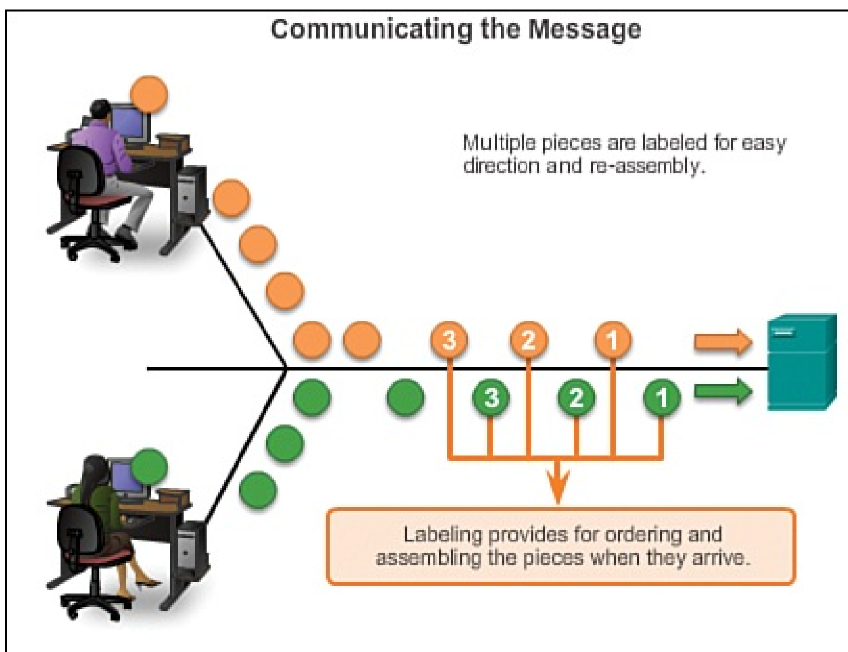
消息要在网络中传输，必须对它进行编码，以特定的格式进行封装，同时需要适当地封装以足够的控制和地址信息，以使它能够从发送方移动到接收方。

消息大小

理论上，视频或邮件信息是能够以大块非中断型流从网络源地址传送到目的地址，但这也意味着同一时刻同一网络其他设备就无法收发消息。这种大型数据流会造成显著延时。并且，如果传输过程中连接断开，整个数据流都会丢失需要全部重传。因此更好的方法是将数据流分割（**segmentation**）为较小的，便于管理的片段，能够带来两点好处：

- 发送较小片段，网络上同时可有多个会话交错进行。这种在网络上将不同会话片段交错进行的过程称为多路传输（**multiplexing**）。
- 分割可提高网络通讯的可靠性。各消息片段从源地址到目的地址无需经过相同路径，如果一条路径被堵塞或断开，其余消息可从替换路径到达目的地址。如果部分消息到不了目的地址，那只需重传丢失部分。

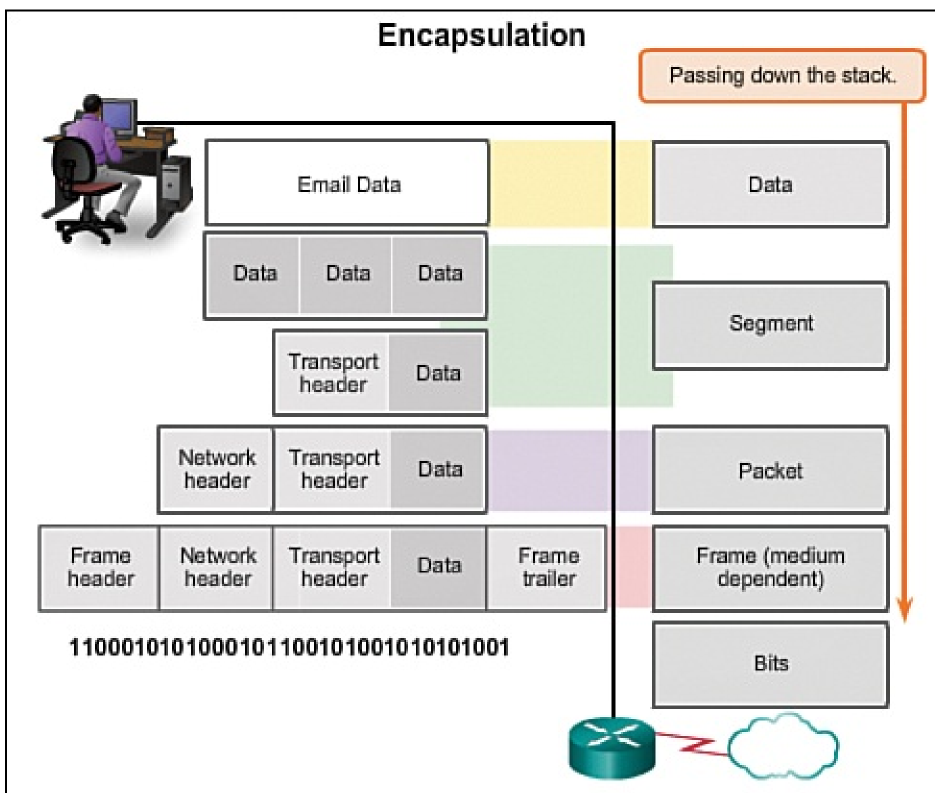
通过对片段打上标签的方式来保证顺序以及在接收时重组。




协议数据单元（**Protocol Data Unit, PDU**）

应用层数据在传输过程中沿着协议栈传递，每一层协议都会向其中添加信息。这就是封装的过程。

数据片段在各层网络结构中采用的形式就称为协议数据单元（**PDU**）。封装过程中，下一层对从上一层收到的PDU进行封装。在处理的每一个阶段PDU都有不同的名字来反应它的功能。




PDU按照TCP/IP协议的命名规范：

- 数据（**Data**）：应用层PDU的常用术语
- 分段（**Segment**）：传输层PDU
- 帧（**Frame**）：网络层PDU
- 比特（**Bits**）：在介质上物理传输数据所使用的PDU。

封装

封装是指在传输之前为数据添加额外的协议头信息的过程。在绝大多数数据通信过程中，源数据在传输前都会封装以数层协议。在网络上发送消息时，主机上的协议栈从上至下进行操作。

以网络服务器为例，HTTP应用层协议发送HTML格式网页数据到传输层，应用层数据被分成TCP分段。各TCP分段被打上标签，称为头（header），表明接收方哪一个进程应当接收此消息。同时也包含使得接收方能够按照原有的格式来重组数据的信息。

传输层将网页HTML数据封装成分段并发送至网络层，执行IP层协议。整个TCP分段封装成IP报文，也就是再添上IP头标签。IP头包括源和目的IP地址，以及发送报文到目的地址所必须的信息。

之后，IP报文发送到接入层，封装以帧头和帧尾。每个帧头都包含源和目的物理地址。物理地址唯一指定了本地网络上的设备。帧尾包含差错校正信息。最后，由服务器网卡将比特编码传输给介质。

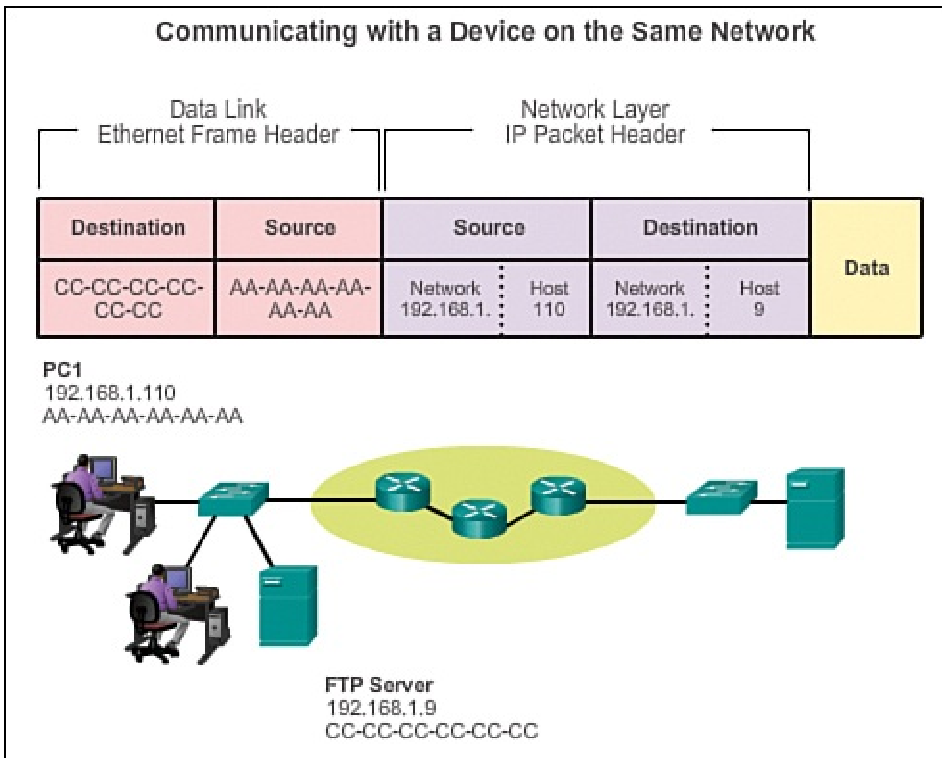
解封装

接收主机以相反的方式进行操作称为解封装。解封装是接收设备移除一层或多层协议头的过程。数据在协议栈中向上移动直到终端应用层伴随着解封装。

访问本地资源：

访问本地网络资源需要两种类型的地址：网络层地址和数据链路层地址。网络层和数据链路层负责将数据从发送设备传输至接收设备。两层协议都有源和目的地址，但两种地址的目的不同。

示例：客户端**PC1**与**FTP**在同一**IP**网络的通信




网络地址

网络层地址或IP地址包含两个部分：网络前缀和主机。路由器使用网络前缀部分将报文转发给适当的网络。最后一个路由器使用主机部分将报文发送给目标设备。同一本地网络中，网络前缀部分是相同的，只有主机设备地址部分不同。

源IP地址：发送设备，即客户端PC1的IP地址：192.168.1.110

目的IP地址：接收设备，即FTP服务器：192.168.1.9

数据链路地址

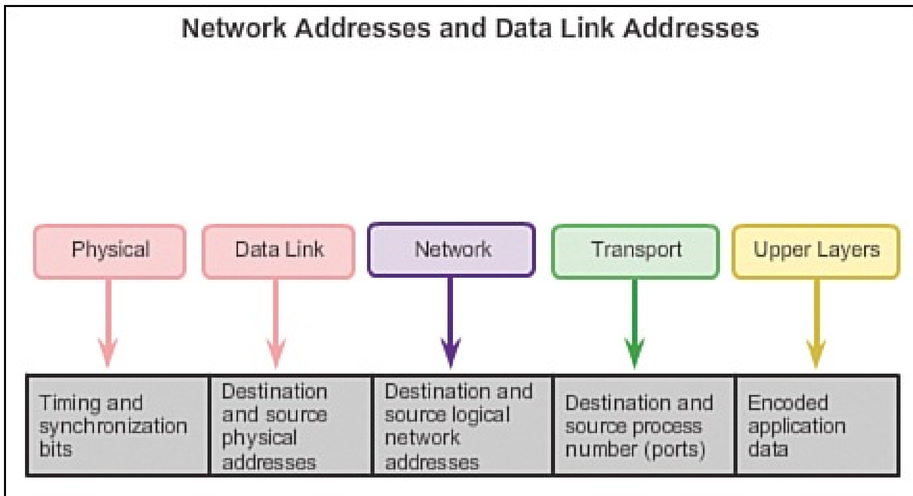
数据链路地址的目的是在同一网络中将数据链路帧从一个网络接口发送至另一个网络接口。以太网LAN和无线网LAN是两种不同物理介质的网络示例，分别有自己的数据链路协议。

当IP报文的发送方和接收方位于同一网络，数据链路帧直接发送到接收设备。以太网上数据链路地址就是以太网MAC地址。MAC地址是物理植入网卡的48比特地址。

源MAC地址：发送IP报文的PC1以太网卡MAC地址，AA-AA-AA-AA-AA-AA。

目的MAC地址：当发送设备与接收设备位于同一网络，即为接收设备的数据链路地址。本例中，FTP MAC地址：CC-CC-CC-CC-CC-CC。

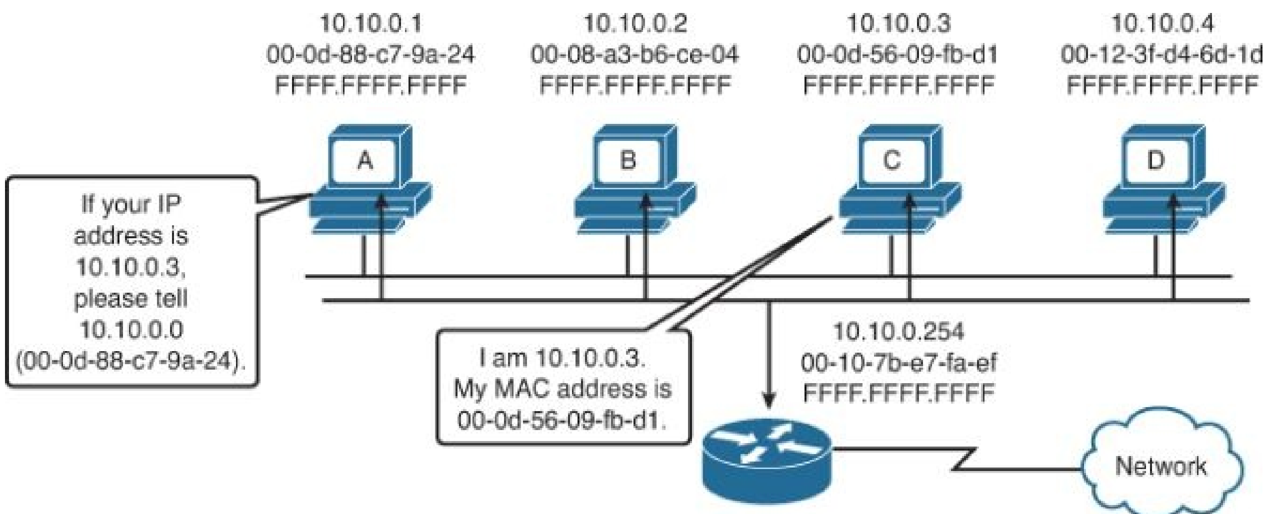
源和目的MAC地址添加到以太网帧中。




**MAC与IP地址**

发送方必须知道接收方的物理和逻辑地址。发送方主机能够以多种方式学习到接收方的IP地址：比如域名系统（Domain Name System, DNS），或通过应用手动输入，如用户指定FTP地址。

以太网MAC地址是怎么识别的呢？发送方主机使用地址解析协议（Address Resolution Protocol, ARP）以检测本地网络的所有MAC地址。如下图所示，发送主机在整个LAN发送ARP请求消息，这是一条广播消息。ARP请求包含目标设备的IP地址，LAN上的每一个设备都会检查该ARP请求，看看是否包含它自身的IP地址。只有符合该IP地址的设备才会发送ARP响应。ARP响应包含ARP请求中IP地址相对应的MAC地址。

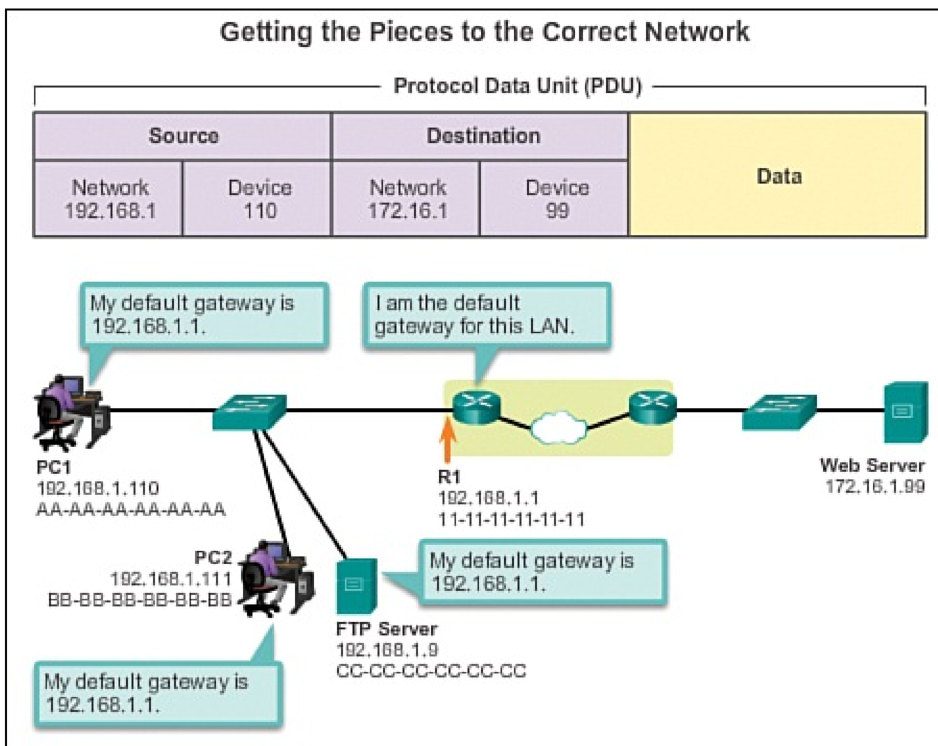


访问远程资源：

默认网关

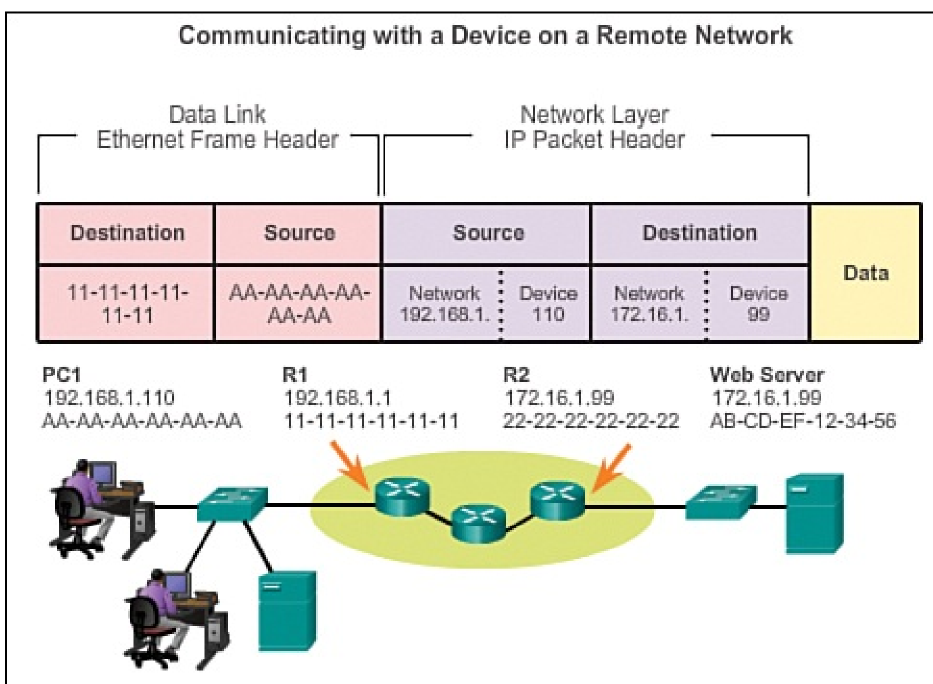
当主机发送消息到远端网络，必须使用路由器，也称为默认网关。默认网关就是位于发送主机同一网络上的路由器的接口IP地址。有一点很重要：本地网络上的所有主机都能够配置自己的默认网关地址。如果该主机的**TCP/IP**设置中没有配置默认网关地址，或指定了错误的默认网关地址，则远端网络消息无法被送达。

如下图所示，LAN上的主机PC 1使用IP地址为192.168.1.1的R1作为默认网关，如果PDU的目的地址位于另一个网络，则主机将PDU发送至路由器上的默认网关。




与远端网络设备通讯

下图显示了客户端主机PC 1与远端IP网络服务器进行通讯的网络层地址与数据链路层地址：




网络地址

当报文的发送方与接收方位于不同网络，源和目的**IP**地址将会代表不同网络上的主机。

源IP地址：发送设备即客户端主机PC 1的IP地址：192.168.1.110。

目的IP地址：接收设备即网络服务器的IP地址：172.16.1.99。

数据链路地址

当报文的发送方与接收方位于不同网络，以太网数据链路帧无法直接被发送到目的主机。以太网帧必须先发送给路由器或默认网关。本例中，默认网关是R1，R1的接口IP地址与PC 1属于同一网络，因此PC 1能够直接达到路由器。

源MAC地址：发送设备即PC 1的MAC地址，PC1的以太网接口MAC地址为：AA-AA-AA-AA-AA-AA。

目的MAC地址：当报文的发送方与接收方位于不同网络，这一值为路由器或默认网关的以太网MAC地址。本例中，即R1的以太网接口MAC地址，即：11-11-11-11-11-11。

IP报文封装成的以太网帧先被传输至R1，R1再转发给目的地址即网络服务器。R1可以转发给另一个路由器，如果目的服务器所在网路连接至R1，则直接发送给服务器。

发送设备如何确定路由器的MAC地址？每一个设备通过自己的TCP/IP设置中的默认网关地址得知路由器的IP地址。之后，它通过ARP来得知默认网关的MAC地址，该MAC地址随后添加到帧中。

# 网络基本功（二）：细说交换机

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

本节介绍交换机的帧转发技术，MAC地址表的维护方式，三种帧转发模式，以及冲突域和广播域。

## 更多信息

帧转发:

网络及电信中的交换概念

以太网上的帧包含源MAC地址与目的MAC地址。交换机从源设备接收到帧并快速发往目的地址。交换的基本概念指基于以下两条准则做出决策的设备：

- 进入（ingress）端口
- 目的地址

术语**ingress**用于描述帧通过特定端口进入设备，**egress**用于描述设备通过特定端口离开设备。交换机做出转发决定的时候，是基于进入端口以及消息的目的地址的。

LAN交换机维护一张表，通过这张表决定如何转发数据流。LAN交换机唯一智能部分是利用这张表基于消息的进入端口和目的地址来转发。一个LAN交换机中只有一张定义了地址和端口的主交换表；因此，无论进入端口如何，同一目的地址的消息永远从同一出口离开。

MAC地址表的动态更新

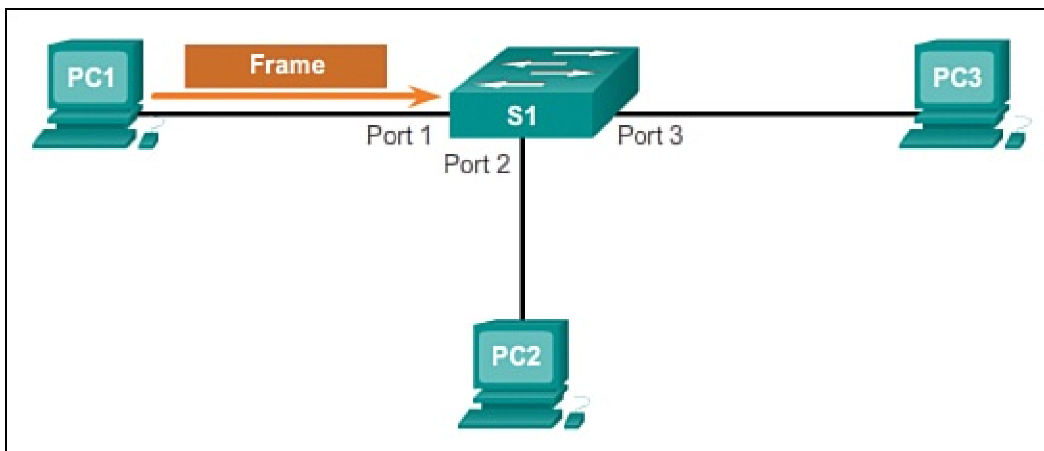
一个交换机要知道使用哪一个端口传送帧，首先必须学习各端口有哪些设备。随着交换机学习到端口与设备的关系，它建立起一张MAC地址表，或内容可寻址寄存表（CAM）。CAM是一种应用于高速查找应用的特定类型的memory。交换机将连接到它的端口的设备的MAC地址记录到MAC表中，然后利用表中信息将帧发送至输出端口设备，该端口已指定给该设备。

记住交换机操作模式的一句简单的话是：交换机学习“源地址”，基于“目的地址”转发。帧进入交换机时，交换机“学习”接收帧的源MAC地址，并将此地址添加到MAC地址表中，或刷新已存在的MAC地址表项的老化寄存器；后续报文如果去往该MAC地址，则可以根据此表项转发。帧转发时，交换机检查目的MAC地址并与MAC地址表中地址进行比较。如果地址在表

中，则转发至表中与MAC地址相对应的端口。如果没有在表中找到目的MAC地址，交换机会转发到除了进入端口以外的所有端口泛洪（flooding）。有多个互连交换机的网络中，MAC地址表对于一个连接至其他交换机的端口记录多个MAC地址。

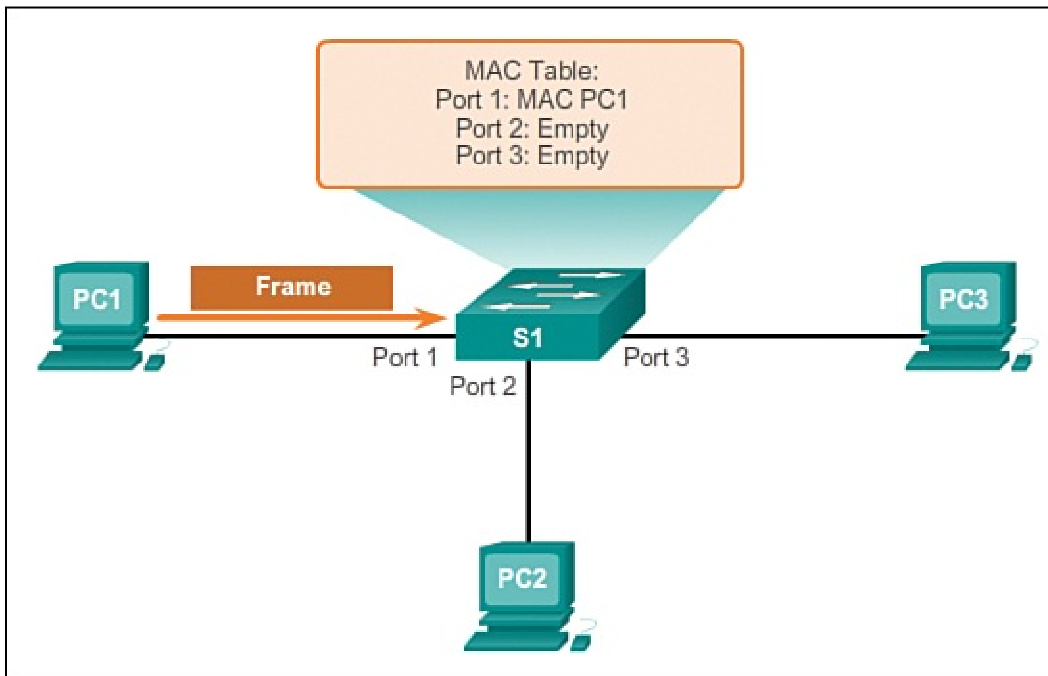
以下步骤描述了更新MAC地址表的方法：

1. 交换机在port 1接收到来自PC 1的帧。




2. 交换机检查源MAC地址并与MAC地址表相比较。

- 如果地址不在表中，则交换机在MAC地址表中将PC 1的源MAC地址关联到进入端口（port 1）。

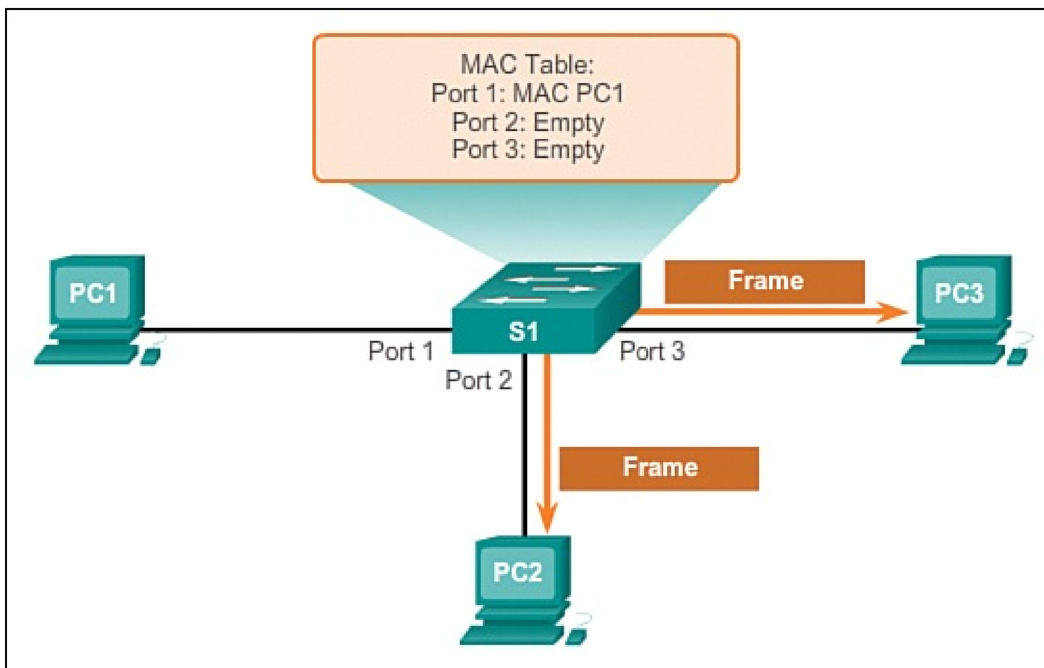



- 如果已经存在该源地址的MAC地址表项，则交换机重置老化计时器。通常一个表项会保持5分钟。

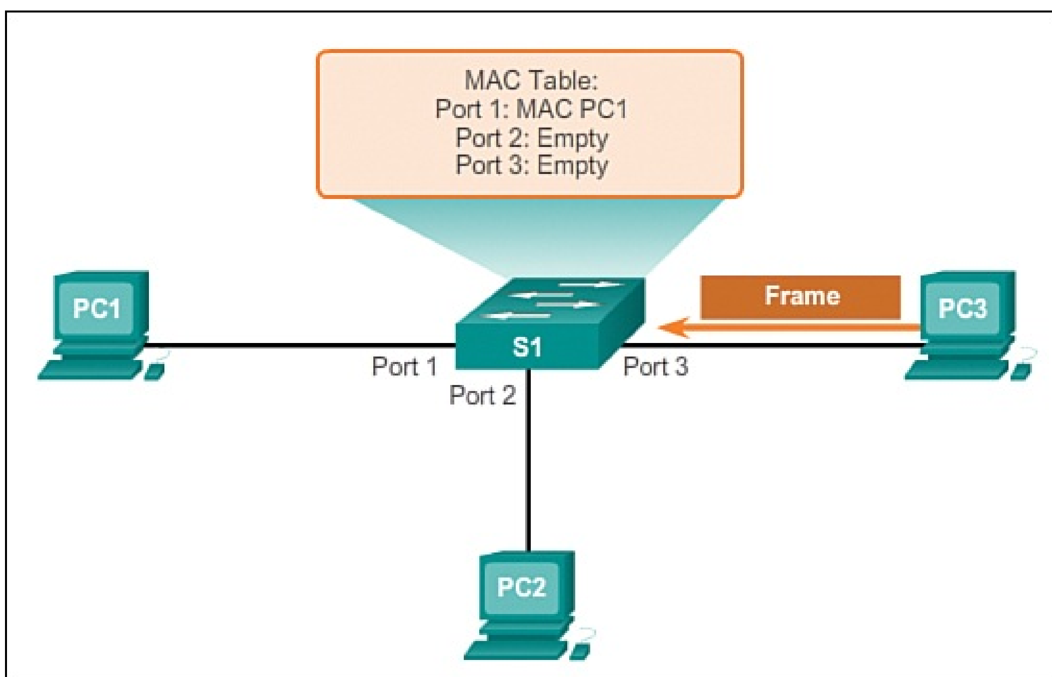
3. 交换机记录源地址信息之后，检查目的地址

- 如果目的MAC地址不在表项中或如果它是一个广播MAC地址，则交换机把该帧泛洪

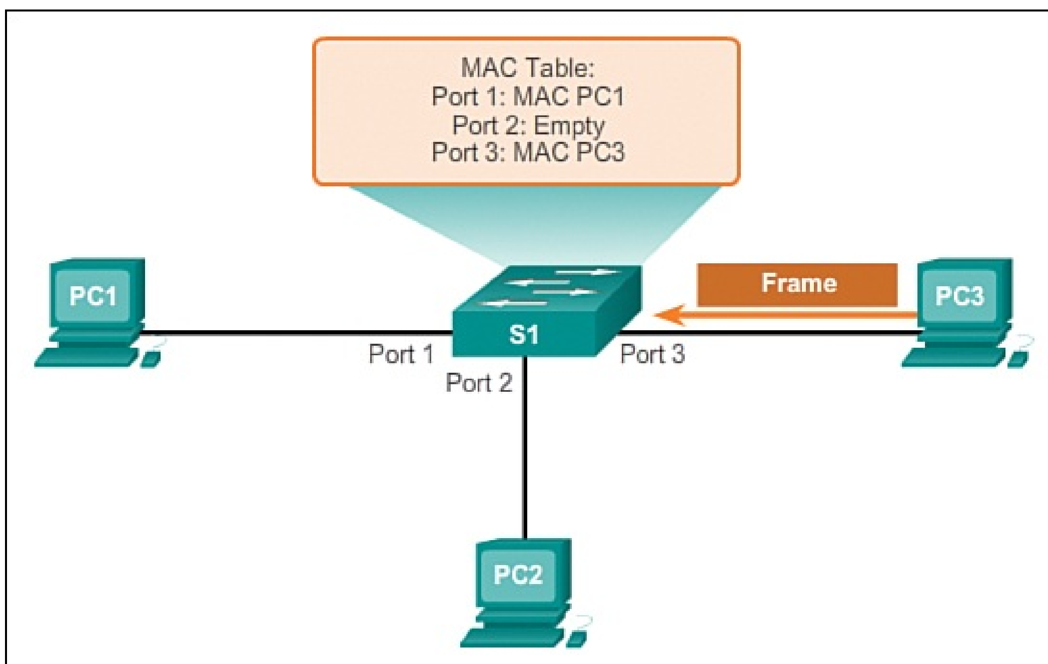
（flood）至除了进入端口以外的所有端口。



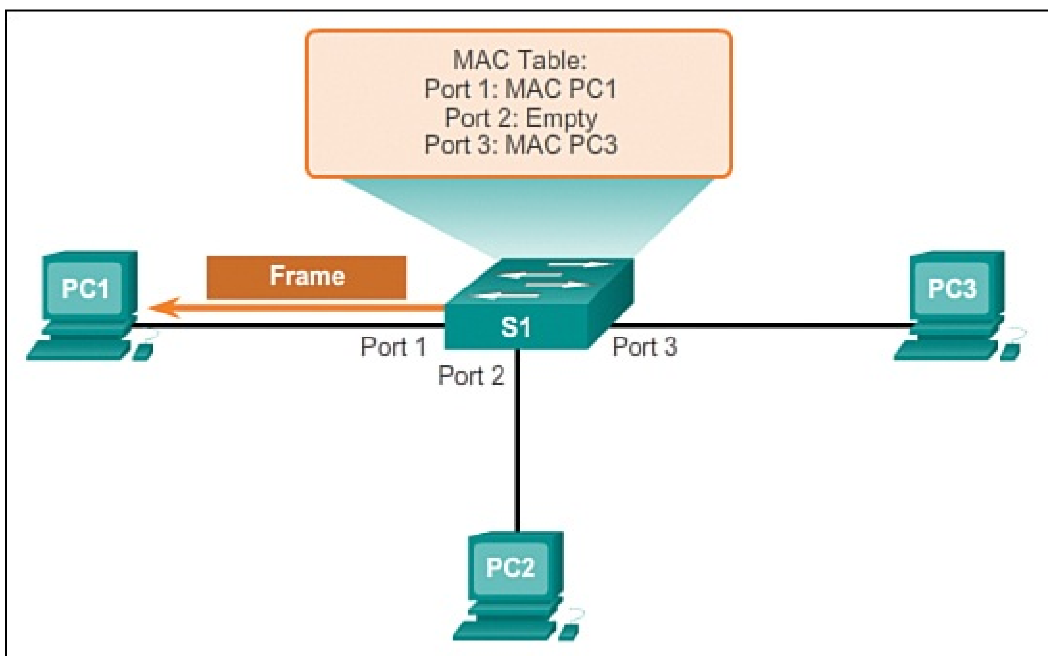

4. 目标设备（PC 3）返回目的地址为PC 1的单播帧。




5. 交换机地址表中输入PC 3的源MAC地址以及进入端口的端口号。在表项中找到该帧的目的地址及关联的输出端口。




6. 交换机现在可以在源和目标设备之间传送帧而无需泛洪，因为地址表中已有指定关联端口的表项。



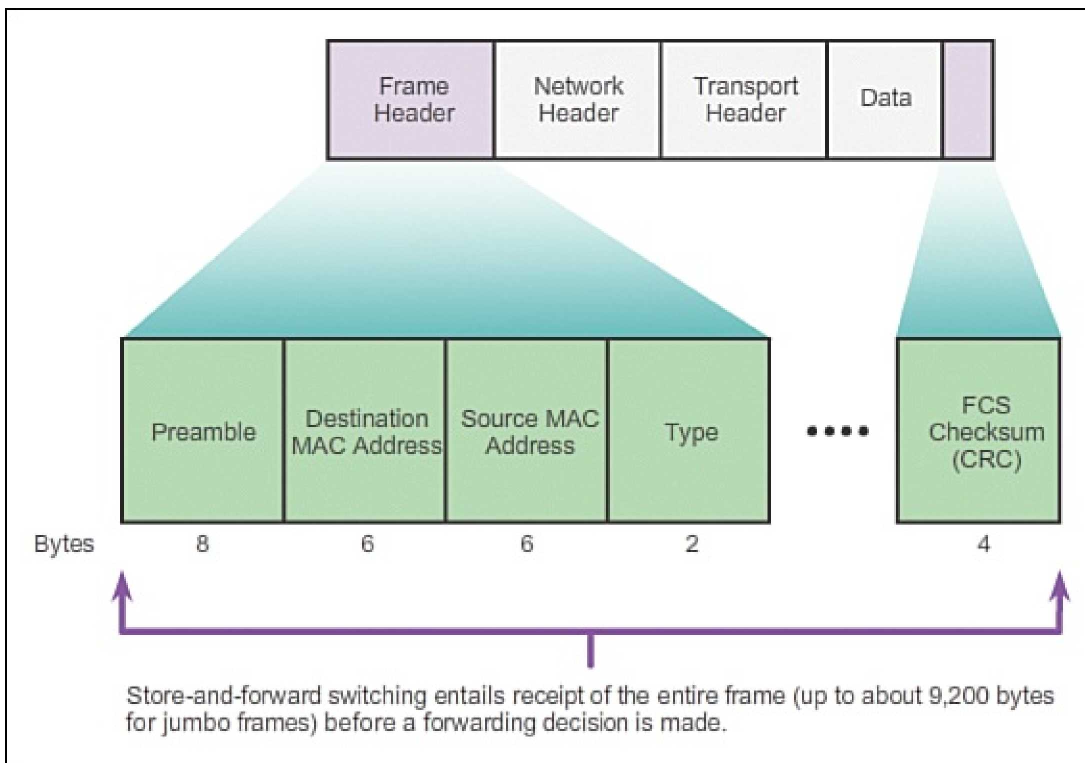

交换机转发方式：

存储转发交换**(Store-and-Forward)**

运行在存储转发模式下的交换机在发送信息前要把整帧数据读入内存并检查其正确性。尽管采用这种方式比采用直通方式更花时间，但采用这种方式可以存储转发数据，从而保证其准确性。由于运行在存储转发模式下的交换机不传播错误数据，因而更适合大型局域网。存储转发模式有两大主要特征区别于直通转发模式：

差错控制：

使用存储转发技术的交换机对进入帧进行差错控制。在进入端口接收完整一帧之后，交换机将数据报最后一个字段的帧校验序列（frame check sequence, FCS）与自己的FCS进行比较。FCS校验过程用以帮助确保帧没有物理及数据链路错误，如果该帧校验正确，则交换机转发。否则，丢弃。




自动缓存：

存储转发交换机通过进入端口缓存，支持不同速率以太网的混合连接。例如，接收到一个以1Gb/s速率发出的帧，转发至百兆以太网端口，就需要使用存储转发方式。当进入与输出端口速率不匹配时，交换机将整帧内容放入缓存中，计算FCS校验，转发至输出缓存之后将帧发出。

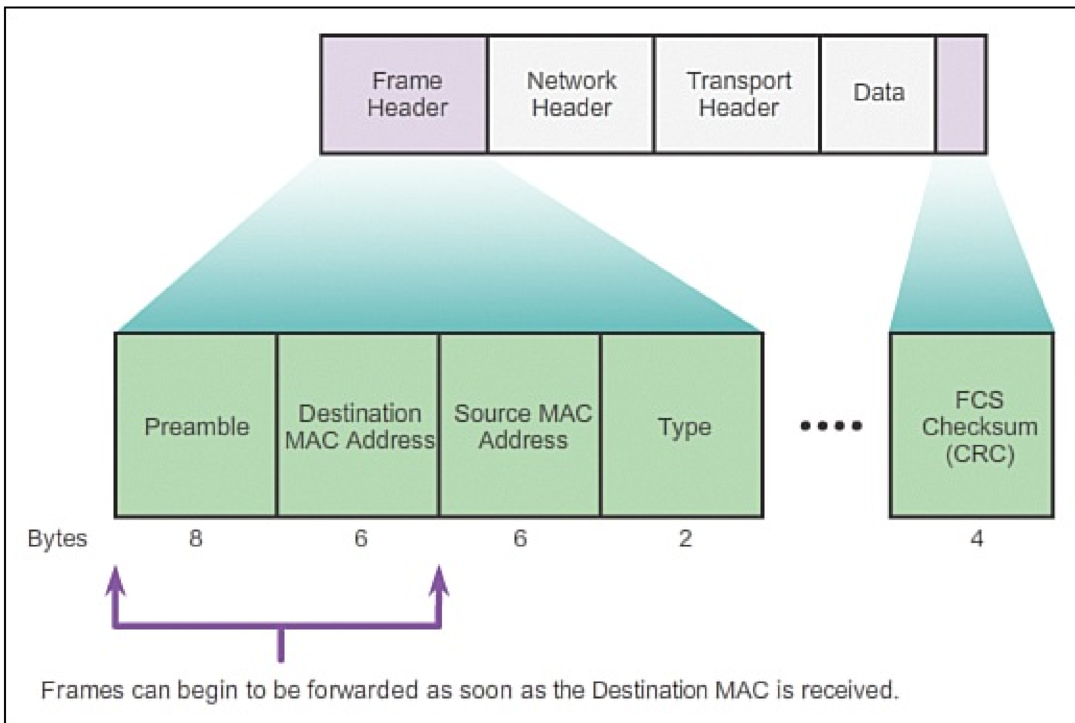
Cisco的主要交换方式是存储转发交换。

直通交换（**Cut-Through**）

直通交换的一个优势是比存储转发技术更为快速。采用直通模式的交换机会在接收完整个数据包之前就读取帧头，并决定把数据发往哪个端口。不用缓存数据也不用检查数据的完整性。这种交换方式有两大特点：快速帧转发以及无效帧处理。

快速帧转发：

如下图所示，一旦交换机在MAC地址表中查找到目的MAC地址，就立刻做出转发决定。而无需等待帧的剩余部分进入端口再做出转发决定。




使用直通方式的交换机能够快速决定是否有必要检查帧头的更多部分，以针对额外的过滤目的。例如，交换机可以检查前14个字节（源MAC地址，目的MAC，以太网类型字段），以及对之后的40字节进行检查，以实现IPv4三层和四层相关功能。

无效帧处理：

对于大多数无效帧，直通方式交换机并不将其丢弃。错误帧被转发至其他网段。如果网络中出现高差错率（无效帧），直通交换可能会对带宽造成不利影响，损坏以及无效帧会造成带宽拥塞。在拥塞情况下，这种交换机必须像存储转发交换机那样缓存。

无碎片转发（**Fragment Free**）

无碎片转发是直通方式的一种改进模式。交换机转发之前检查帧是否大于64字节（小于则丢弃），以保证没有碎片帧。无碎片方式比直通方式拥有更好的差错检测，而实际上没有增加延时。它比较适合于高性能计算应用，即进程到进程延时小于10毫秒的应用场景。

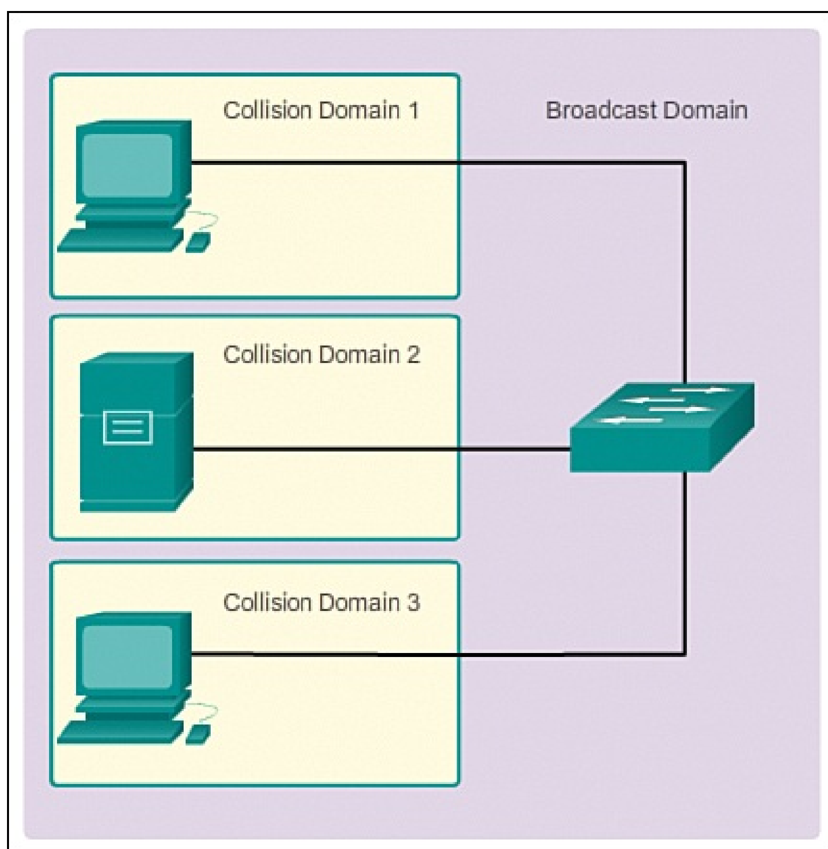
交换机域：

交换机比较容易混淆的两个术语是冲突域和广播域。这一段讲述这两个影响LAN性能的重要概念。

冲突域

设备间共享同一网段称为冲突域。因为该网段内两个以上设备同时尝试通讯时，可能发生冲突。使用工作在数据链路层的交换机可将各个网段的冲突域隔离，并减少竞争带宽的设备数量。交换机的每一个端口就是一个新的网段，因为插入端口的设备之间无需竞争。结果是每一个端口都代表一个新的冲突域。网段上的设备可以使用更多带宽，冲突域内的冲突不会影响到其他网段，这也成为微网段。

如下图所示，每一个交换机端口连接到一台主机，每一个交换机端口代表一个隔离的冲突域。




广播域

尽管交换机按照MAC地址过滤大多数帧，它们并不能过滤广播帧。LAN上的交换机接收到广播包后，必须对所有端口泛洪。互连的交换机集合形成了一个广播域。网络层设备如路由器，可隔离二层广播域。路由器可同时隔离冲突和广播域。

当设备发出二层广播包，帧中的目的MAC地址被设置为全二进制数，广播域中的所有设备都会接收到该帧。二层广播域也称为MAC广播域。MAC广播域包含LAN上所有接收到广播帧的设备。广播通信比较多时，可能会带来广播风暴。特别是在包含不同速率的网段，高速网段产生的广播流量可能导致低速网段严重拥挤，乃至崩溃。

# 网络基本功（三）：细说VLAN与Trunk

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



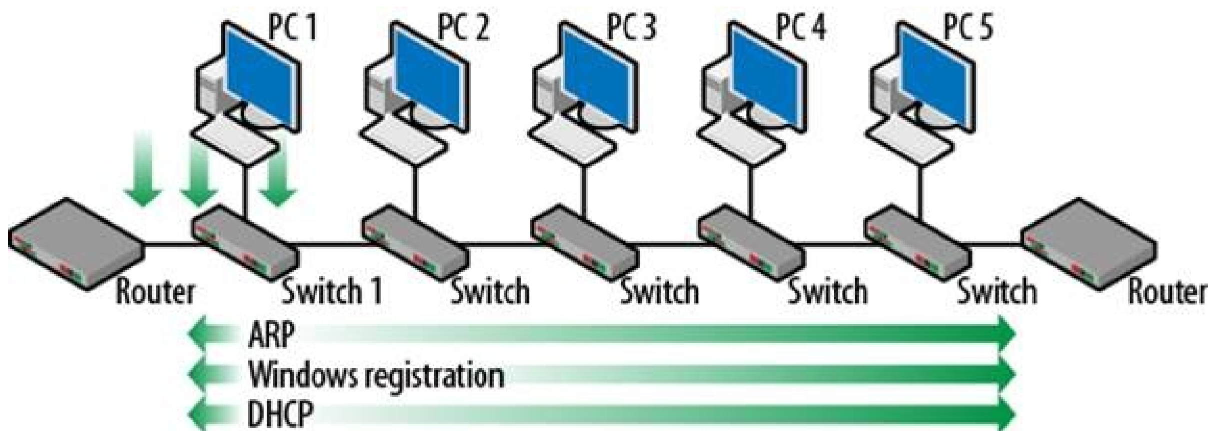

## 介绍

网络性能是影响业务效率的一个重要因素。将大型广播域分段是提高网络性能的方法之一。路由器能够将广播包阻隔在一个接口上，但是，路由器的LAN接口数量有限，它的主要功能是在网络间传输数据，而不是对终端设备提供网络接入。访问LAN的功能还是由接入层交换机来实现。与三层交换机相类似，通过在二层交换机上创建VLAN来减少广播域。现代交换机就是通过VLAN来构造的，因此在某种程度上，学习交换机就是学习VLAN。

## 更多信息

问题的产生**:**

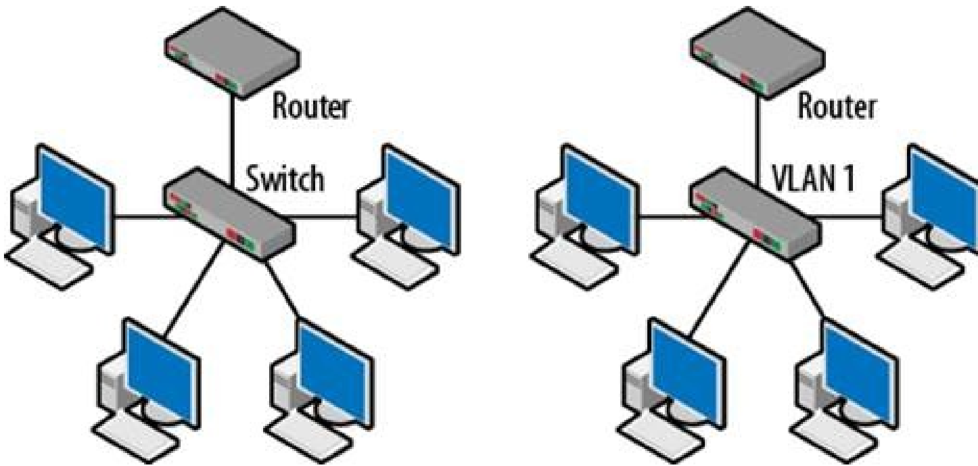
上一节已经讲过广播域的概念：即广播帧传播覆盖的范围。如下图所示，当网络上的所有设备在广播域产生大量的广播以及多播帧，就会与数据流竞争带宽。这是由网络管理数据流组成，如：ARP，DHCP，STP等。如下图所示，假设PC 1产生ARP，Windows登录，DHCP等请求：




这些广播帧到达交换机1之后，遍历整个网络并到达所有节点直至路由器。随着网络节点增加，开销的总数也在增长，直至影响交换机性能。通过实施VLAN断开广播域将数据流隔离开来，能够解决这一问题。

什么是**VLAN**：

VLAN（virtual local area network）是一组与位置无关的逻辑端口。VLAN就相当于一个独立的三层网络。VLAN的成员无需局限于同一交换机的顺序或偶数端口。下图显示了一个常规的部署，左边这张图节点连接到交换机，交换机连接到路由器。所有的节点都位于同一IP网络，因为他们都连接到路由器同一接口。



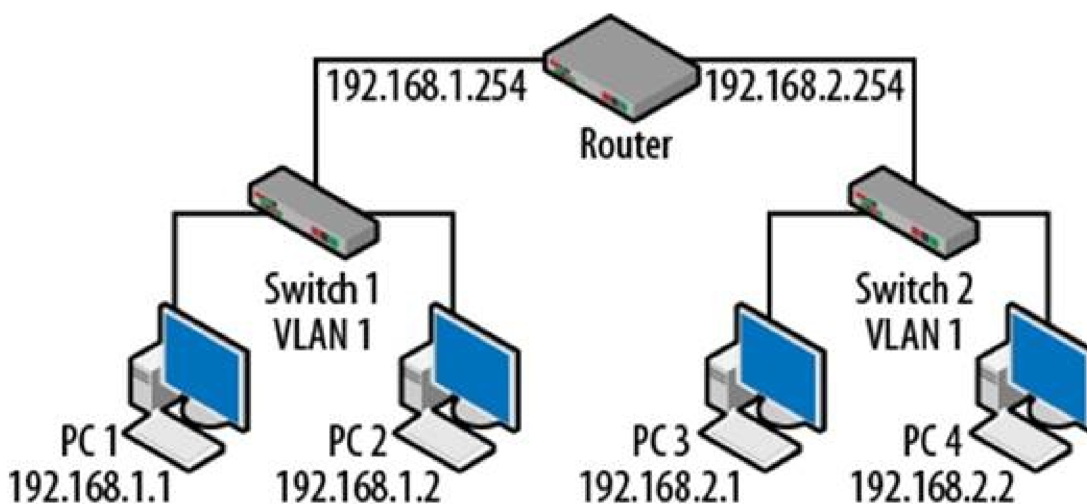

图中没有显示的是，缺省情况下，所有节点实际上都是同一VLAN。因此，这种拓扑接口可看作是基于同一VLAN的，如上面右图所示。例如，Cisco设备默认VLAN是VLAN 1，也称为管理VLAN。默认配置下包含所有的端口，体现在源地址表（source address table，SAT）中。该表用于交换机按照目的MAC地址将帧转发至合适的二层端口。引入VLAN之后，源地址表按照VLAN将端口与MAC地址相对应起来，从而使得交换机能够做出更多高级转发决策。下图显示了show mac address table和show vlan命令的显示输出。所有端口（FA0/1 – FA0/24）都在VLAN 1。

```
Switch#sh mac-address-table
          Mac Address Table
-------------------------------------------

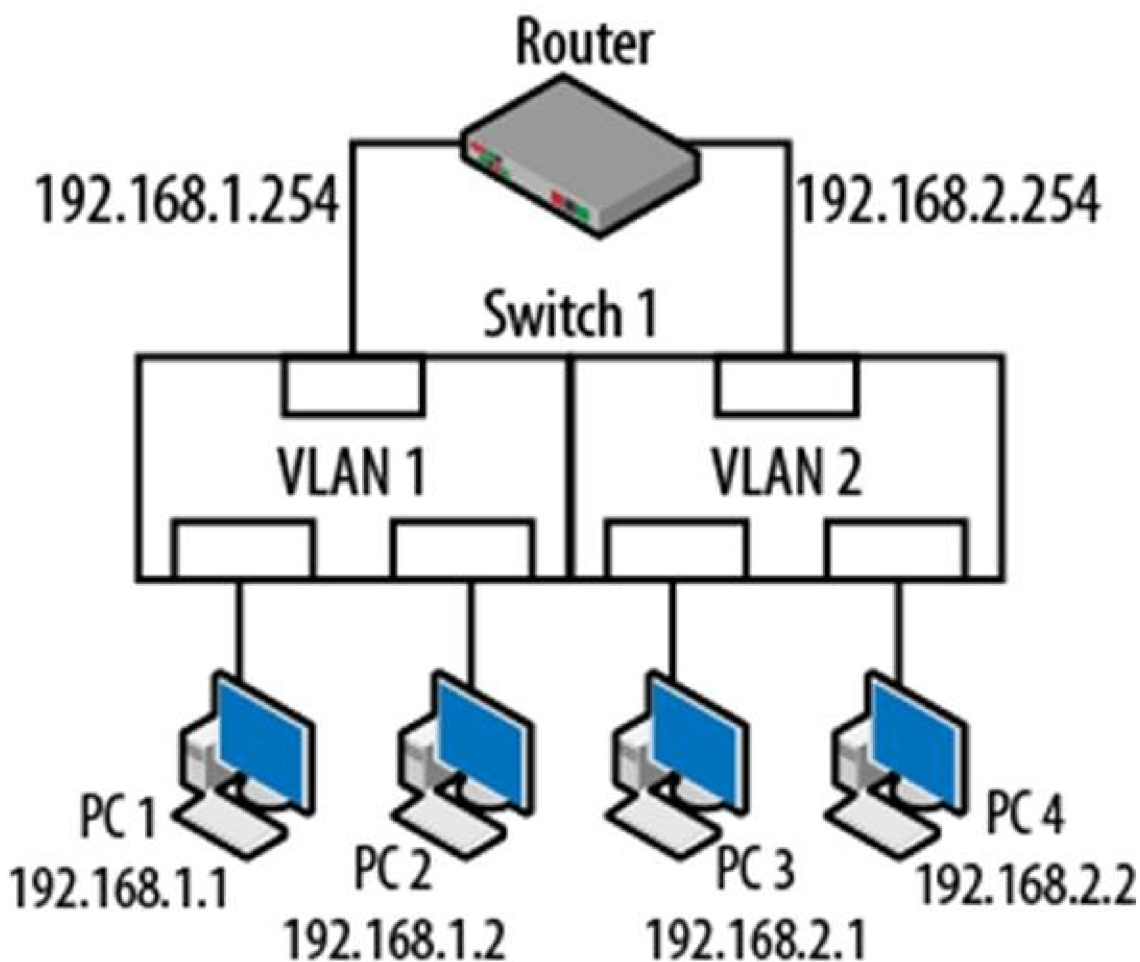
Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
 All    000c.85aa.ea40    STATIC      CPU
 All    0100.0ccc.cccc    STATIC      CPU
 All    0100.0ccc.cccd    STATIC      CPU
 All    0100.0cdd.dddd    STATIC      CPU
   1    000a.f458.6c58    DYNAMIC     Fa0/24
   1    0013.f7d1.de9b    DYNAMIC     Fa0/24
   1    0013.f7d1.e016    DYNAMIC     Fa0/1
   1    0013.f7d2.06d5    DYNAMIC     Fa0/24
   1    0013.f7d2.06e1    DYNAMIC     Fa0/2
Total Mac Addresses for this criterion: 9
Switch#
Switch#show vlan

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23
```

另一种常用的拓扑结构是两个交换机被一个路由器分离开来，如下图所示。这种情况下，每台交换机各连接一组节点。每个交换机上的各节点共享一个IP地址域，这里有两个网段：192.168.1.0和192.168.2.0。




注意到两台交换机的VLAN相同。非本地网络数据流必须经过路由器转发。路由器不会转发二层单播，多播以及广播帧。这种拓扑逻辑在两个地方类似于多VLAN：同一VLAN下的节点共享一个通用地址域，非本地数据流（对应多VLAN情况不同VLAN的节点）需通过路由器转发。在一台交换机上添加一个VLAN，去掉另一台交换机的话，结构如下所示：

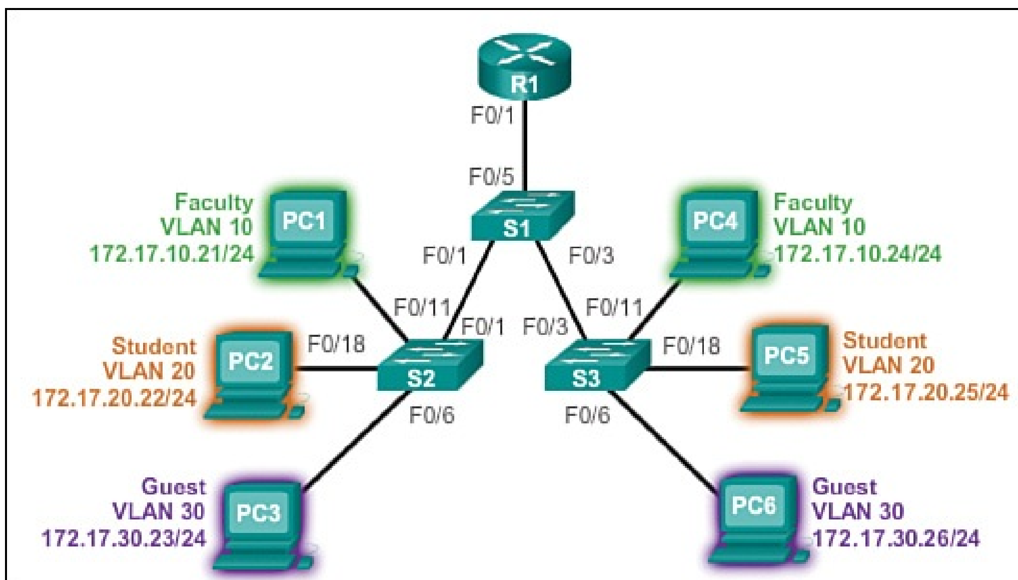



每一个VLAN相当于一个独立的三层IP网络，因此，192.168.1.0上的节点试图与192.168.2.0上的节点通信时，不同**VLAN**通信必须通过路由器，即使所有设备都连接到同一交换机。二层单播，多播和广播数据只会在同一VLAN内转发及泛洪，因此VLAN 1产生的数据不会为VLAN

2节点所见。只有交换机能看得到VLAN，节点和路由器都感觉不到VLAN的存在。添加了路由决策之后，可以利用3层的功能来实现更多的安全设定，更多流量以及负载均衡。

**VLAN**的作用：

安全性：每一个分组的敏感数据需要与网络其他部分隔离开，减少保密信息遭到破坏的可能性。如下图所示，VLAN 10上的教职工主机完全与学生和访客数据隔离。




节约成本：无需昂贵的网络升级，并且带宽及上行链路利用率更加有效。

性能提高：将二层网络划分成多个逻辑工作组（广播域）减少网络间不必要的数据流并提升性能。

缩小广播域：减少一个广播域上的设备数量。如上图所示：网络上有六台主机但有三个广播域：教职工，学生，访客。

提升IT管理效率：网络需求相似的用户共享同一VLAN，从而网络管理更为简单。当添加一个新的交换机，在指定端口VLAN时，所有策略和步骤已配置好。

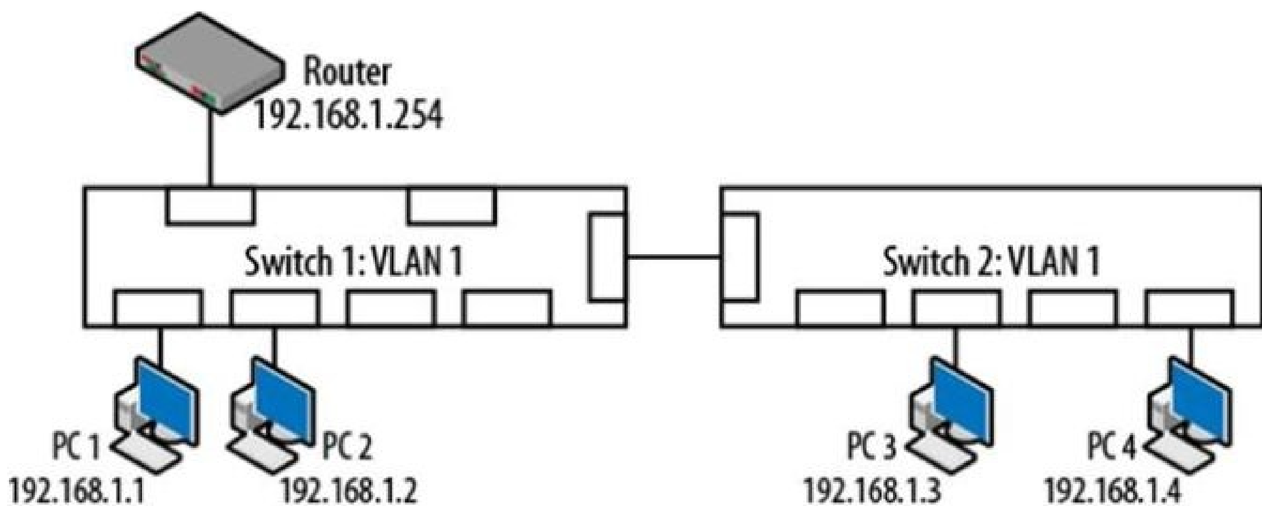
简化项目和应用管理：VLAN将用户和网络设备汇集起来，以支持不同的业务或地理位置需求。

每一个VLAN对应于一个IP网络，因此，部署VLAN的时候必须结合考虑网络地址层级的实现情况。

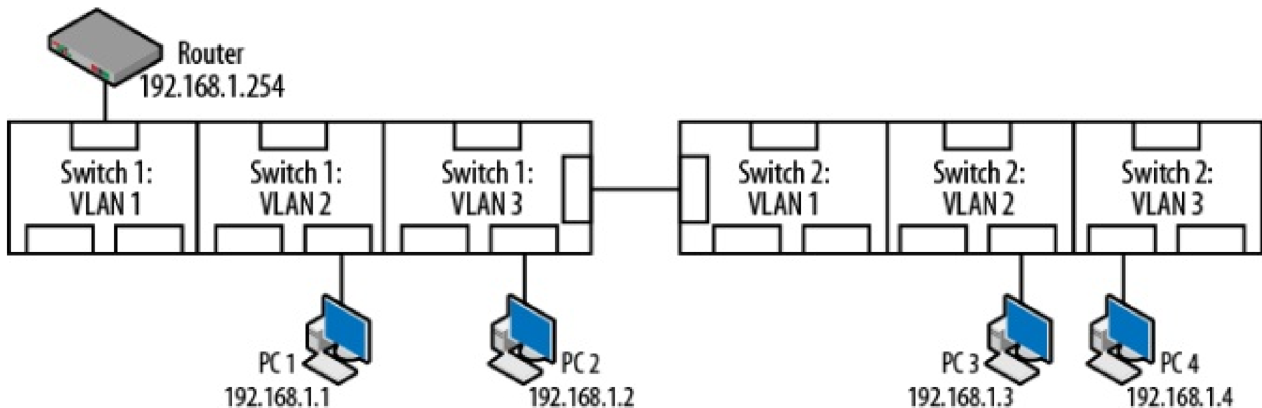
交换机间**VLAN**：

多交换机的情况下，VLAN是怎么工作的呢？下图所示的这种情况，两个交换机VLAN相同，都是默认VLAN 1，即两个交换机之间的联系同在VLAN 1之内。路由器是所有节点的出口。

这时单播，多播和广播数据自由传输，所有节点属于同一IP地址。这时节点之间的通信不会有问题，因为交换机的SAT显示它们在同一VLAN。

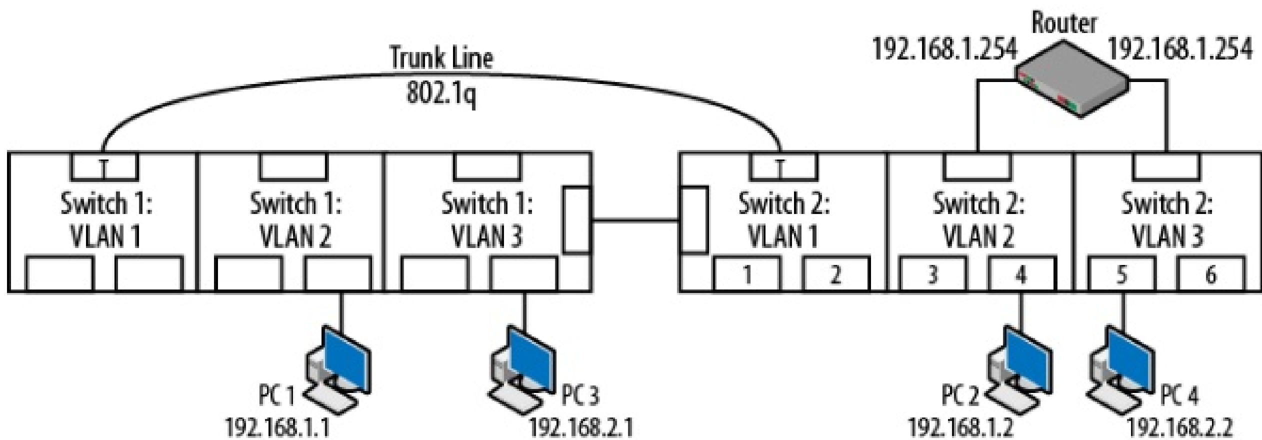



而下面这种连接方式就会有问题。由于VLAN在连接端口的主机之间创建了三层边界，它们将无法通信。




仔细看上图，这里有很多问题。第一，所有主机都在同一IP网，尽管连接到不同的VLAN。第二，路由器在VLAN 1,因此与所有节点隔离。最后，两台交换机通过不同的VLAN互连。每一点都会造成通信阻碍，合在一起，网络各元素之间会完全无法通信。

交换机用满或同一管理单元物理上彼此分离的情形是很常见的。这种情况下，VLAN需要通过trunk延伸至相邻交换机。trunk能够连接交换机，在网络间传载VLAN信息。如下图所示：




对之前的拓扑的改进包括：

- PC 1和PC 2分配到192.168.1.0网段以及VLAN 2。

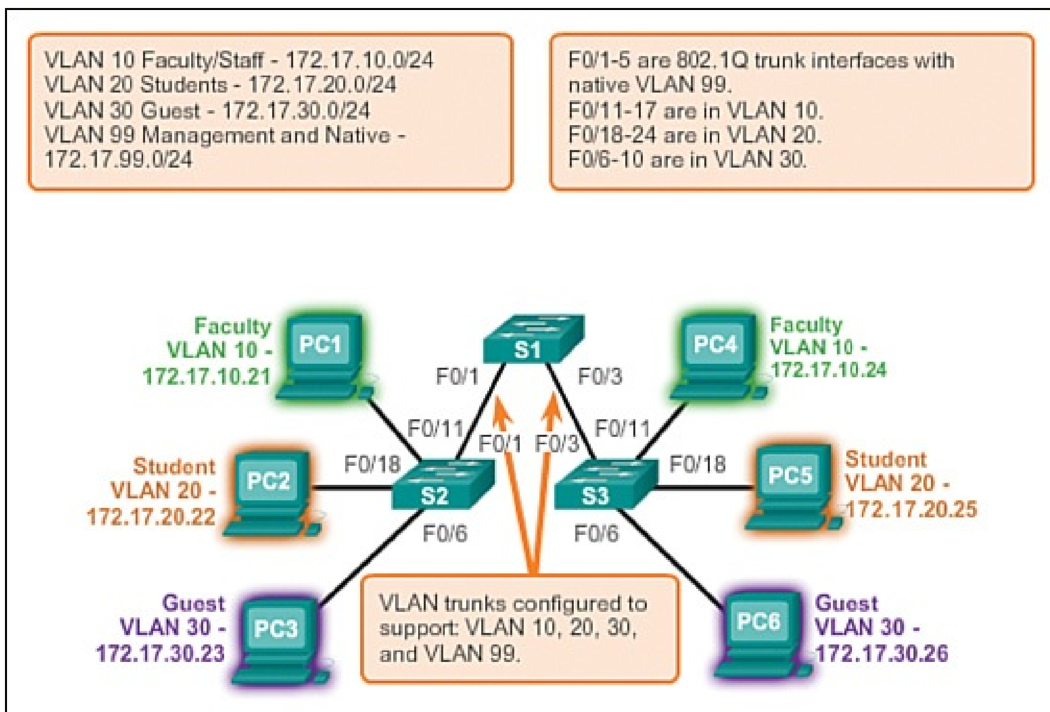
- PC 3和PC 4分配到192.168.2.0网段以及VLAN 3。
- 路由器接口连接到VLAN 2和VLAN 3。
- 交换机间通过trunk线互连。

注意到trunk端口出现在VLAN 1，他们没有用字母T来标识。trunk在任何VLAN都没有成员。现在VLAN跨越多交换机，同一VLAN下的节点可以物理上位于任何地方。

什么是**Trunk**：

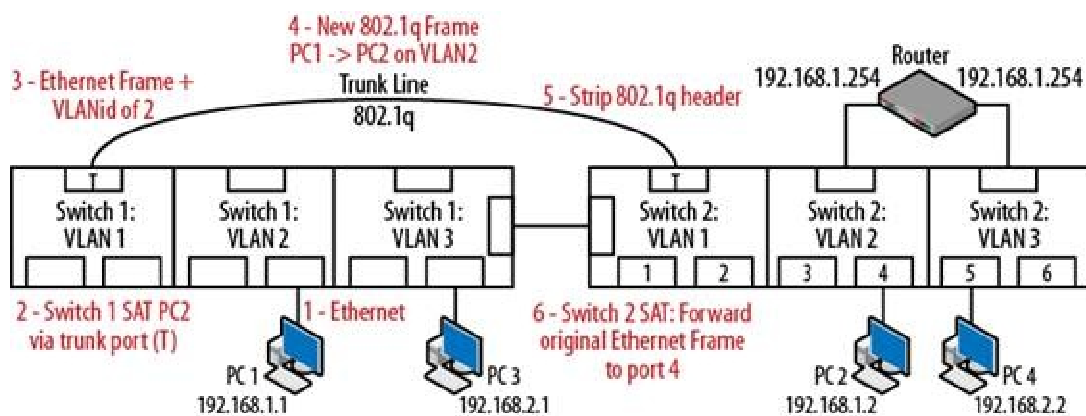
Trunk是在两个网络设备之间承载多于一种VLAN的端到端的连接，将VLAN延伸至整个网络。没有VLAN Trunk，VLAN也不会非常有用。VLAN Trunk允许VLAN数据流在交换机间传输，所以设备在同一VLAN，但连接到不同交换机，能够不通过路由器来进行通信。

一个**VLAN trunk**不属于某一特定**VLAN**，而是交换机和路由器间多个VLAN的通道。如下图所示，交换机S1和S2，以及S1和S3之间的链路，配置为传输从VLAN10,20,30以及90的数据流。该网络没有VLAN trunk就无法工作。




当安装好trunk线之后，帧在trunk线传输是就可以使用trunk协议来修改以太网帧。这也意味着交换机端口有不止一种操作模式。缺省情况下，所有端口都称为接入端口。当一个端口用于交换机间互连传输VLAN信息时，这种端口模式改变为trunk，节点也路由器通常不知道VLAN的存在并使用标准以太网帧或“untagged”帧。trunk线能够使用“tagged”帧来标记VLAN或优先级。

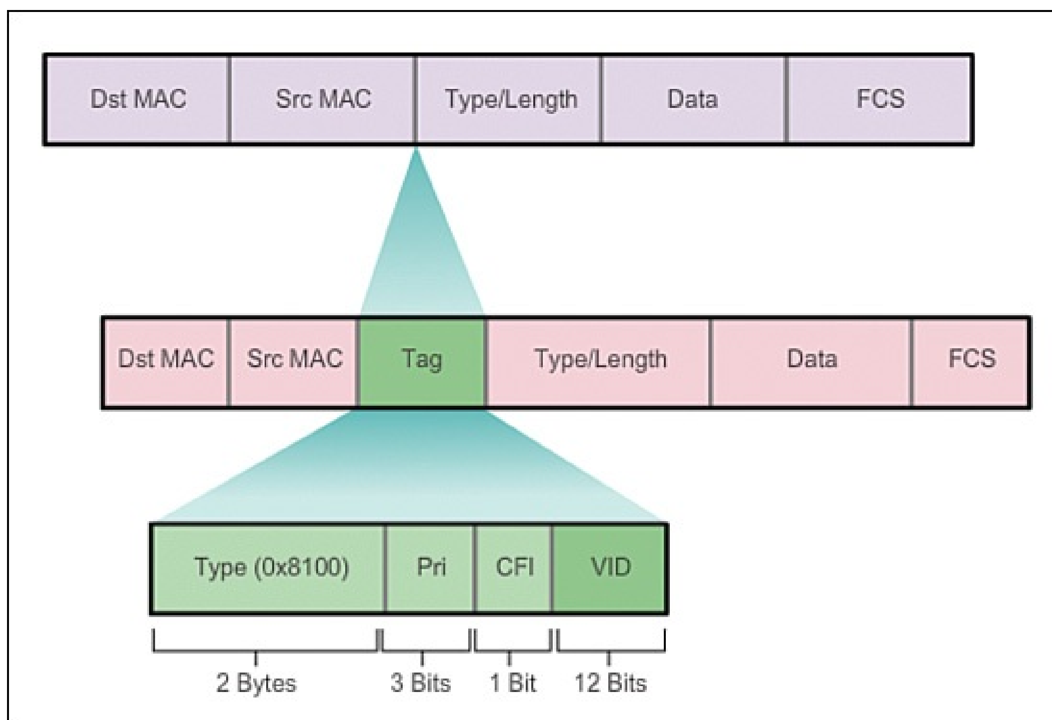
因此，在trunk端口，运行trunk协议来允许帧中包含trunk信息。如下图所示：




PC 1在经过路由表处理后向PC 2发送数据流。这两个节点在同一VLAN但不同交换机。步骤如下：

- 以太网帧离开PC 1到达Switch 1。
- Switch 1的SAT表明目的地是trunk线的另一端。
- Switch 1使用trunk协议在以太网帧中添加VLAN id。
- 新帧离开Switch 1的trunk端口被Switch 2接收。
- Switch 2读取trunk id并解析trunk协议。
- 源帧按照Switch 2的SAT转发至目的地（端口4）。

VLAN tag如下图所示，包含类型域，优先级域，CFI（Canonical Format Indicator）指示MAC数据域，VLAN ID。



# 网络基本功（四）：细说路由（上）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

以太网交换机工作在第二层即数据链路层，用于在同一网络内部转发以太网帧。但是，当源和目的IP地址位于不同网络时，以太网帧必须发送给路由器。路由器负责在不同网络间传输报文，通过路由表来决定最佳转发路径。当主机将报文发送至不同IP地址时，由于主机无法直接与本地网络以外的设备通信，报文被转发至默认网关。默认网关就是数据流从本地网络路由至远端设备的目的地。它通常用来连接本地网与公共网。

## 更多信息

报文转发过程:

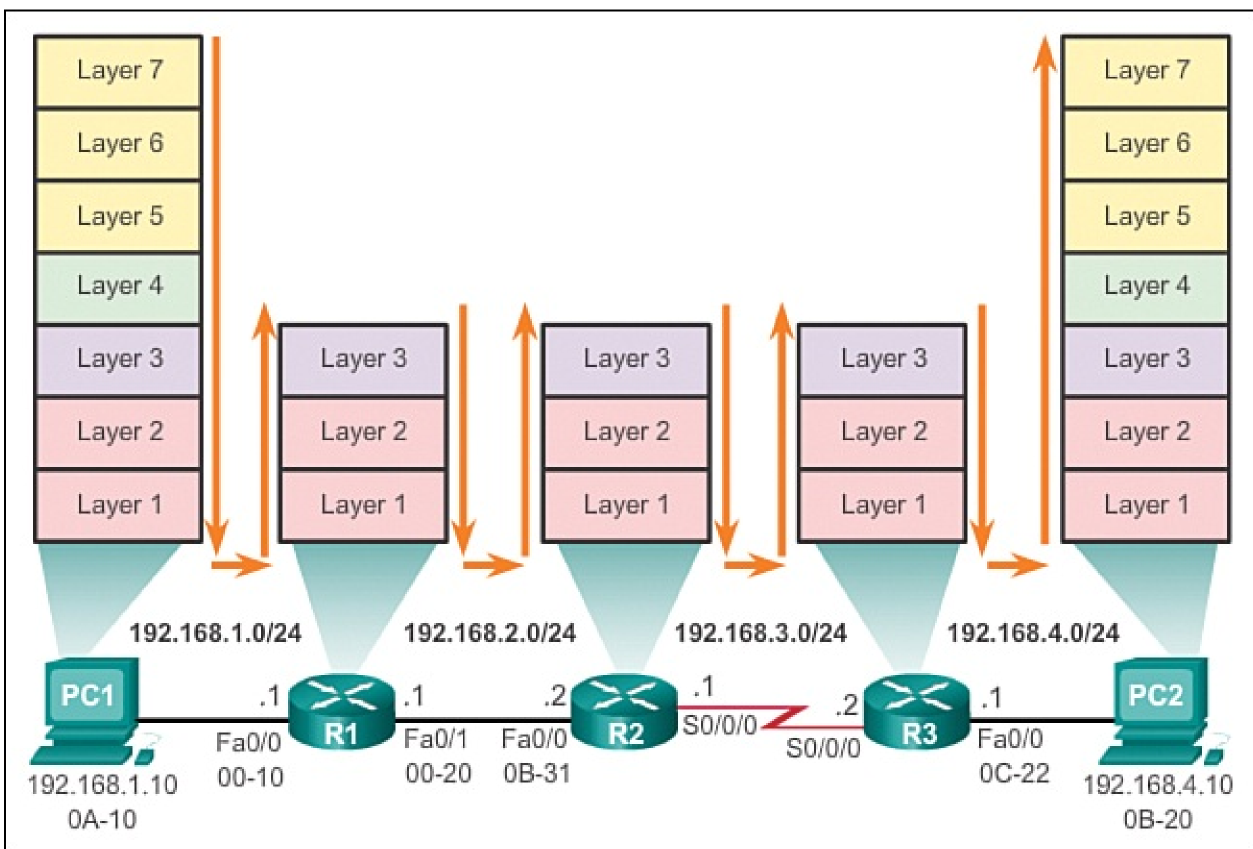
路由器在一个接口接收报文并将它从另一个接口转发出去，这一过程的关键步骤是为输出链路将报文封装在适当的数据链路帧中。路由器主要执行以下三个步骤：

1. 将第二层的帧头和帧尾移除，解析出第三层报文。

2. 检查IP报文的目的IP地址，在路由表中查找最佳路由。

3. 如果路由器找到一条最佳路径，则将三层报文封装到新的二层帧中，并将帧转发到输出端口。

如下图所示：设备有三层IPv4地址，以太网接口有二层数据链路地址。例如PC 1的IPv4地址192.168.1.10，示例MAC地址0A-10。在报文从原设备传输至目的设备的过程中，三层**IP**地址不会改变。但是，每一跳随着报文在路由器中被解封装和重新封装，二层数据链路地址都会改变。很可能报文被封装成与接收时不同的另一种类型的二层帧。




发送报文：

PC 1发送报文给PC 2时，首先必须确定目的IPv4地址是否位于同一网络。PC 1通过将自己的IPv4地址与子网掩码做与操作，来判断PC 1所属的网段。接下来，PC 1对目的IPv4地址与PC 1的子网掩码做同样的与操作。如果目的网络地址与PC 1网络相同，则PC 1不使用默认网关，而是在ARP缓存中查找目的IPv4地址的设备MAC地址。如果MAC地址不在缓存中，则PC 1产生一个ARP请求来获取地址并将报文发给目的地址。如果目的网络地址位于另一网络，则PC 1将报文转发至默认网关。

要确定默认网关的MAC地址，PC 1在它的ARP表中查找默认网关的IPv4地址以及相应的MAC地址。如果ARP表中没有默认网关的对应表项，则PC 1发送ARP请求。路由器R1回复ARP响应。之后PC 1将报文转发至默认网关的MAC地址，即路由器R1的Fa0/0接口。

转发至下一跳：

R1从PC 1接收到以太网帧后执行以下步骤：

1. R1检查目的MAC地址，与接收端口FastEthernet 0/0相匹配，因此，将帧复制到buffer。

2. R1识别以太网类型为0x800，意味着以太网帧的数据部分包含IPv4报文。

3. R1解封装该以太网帧。

4. 由于目的IPv4地址与R1直连的任何网络都不相符，R1在路由表中查找包含该目的IPv4地址主机的网络地址。本例中，路由表中有192.168.4.0/24网络的路由。目的IPv4地址为192.168.4.10，即该网络上的主机IPv4地址。

R1找到192.168.4.0/24路由的下一条IPv4地址为192.168.2.2以及输出端口FastEthernet 0/1，这意味着IPv4报文封装到一个新的以太网帧中，目标MAC地址是下一跳路由器的MAC地址。

由于下一个接口是在以太网上，所以R1必须用ARP解析出下一跳IPv4地址的MAC地址。

1. R1在ARP cache中查找下一跳IPv4地址192.168.2.2。如果表项不在ARP cache中，R1会从FastEthernet 0/1 接口发送ARP请求，R2会返回ARP响应。R1之后在ARP cache中更新192.168.2.2的MAC地址。

2. IPv4报文封装到新的以太网帧中并从R1的FastEthernet 0/1 接口转发出去。

到达目的地：

当帧到达R3时执行以下步骤：

1. R3将数据链路帧复制到它的buffer。

2. R3解封装该数据链路帧。

3. R3在路由表中查找该目的IPv4地址。R3路由表中有直接连接到该网络的路由。这表示报文可直接发送到目的设备而无需发送至路由器。

由于输出接口是一个直连以太网，所以R3必须用ARP解析出目的IPv4地址的MAC地址。

1. R3在它的ARP cache中查找目的IPv4地址，如果此ARP cache中没有此表项，R3会从FastEthernet 0/0 接口发送ARP请求。PC 2回复ARP响应告知它的MAC地址。R3之后在ARP cache中更新192.168.4.10的MAC地址。

2. IPv4报文封装到新的以太网帧中并从R3的FastEthernet 0/0 接口发出。

3. 当PC 2接收到该帧，检查帧的目的MAC地址，与网卡接收端口的MAC地址相匹配，PC 2于是将帧的剩余部分复制到自己的buffer。

4. PC 2识别到以太网类型为0x800，也就是帧的数据部分包含IPv4报文。

5. PC 2解封装以太网帧，将IPv4报文传递给操作系统的IPv4进程。

路由表:

路由表存储的信息包括：

直连路径：来自活动路由接口的路径。当接口为活动状态并配置了IP地址时，路由器添加一条直连路径。

远端路径：远端的网络连接到其他路由。通过静态配置或动态路由协议到达该网络。

路由表是存储在RAM中的一份数据文件，用于存储直连以及远端网络的路由信息。路由表中包含网络或下一跳地址的信息。这些信息告知路由器可以通过将报文发送至代表下一跳地址的路由器以最佳路劲到达目的地址。下一跳信息也可以是到下一个目的地的输出接口。

路由表内容：

Cisco IOS路由器可用**show IP route**命令显示IPv4路由表。路由器还提供一些额外的路由信息，包括路径是怎样学习到的，路径在表中有多长时间，使用哪一接口去到达预定义的目的地。

路由表中的表项可作为以下内容添加：

本地路径接口：当接口配置并激活时添加。

直连接口：当接口配置并激活时添加。

静态路径：当手动配置路径并且输出接口激活时。

动态路由协议：当路由协议动态学习到网络时添加，如EIGRP或OSPF。

路由表项的来源通过代码来标识，代码表明路径是怎样学习到的。例如，常用代码包括：

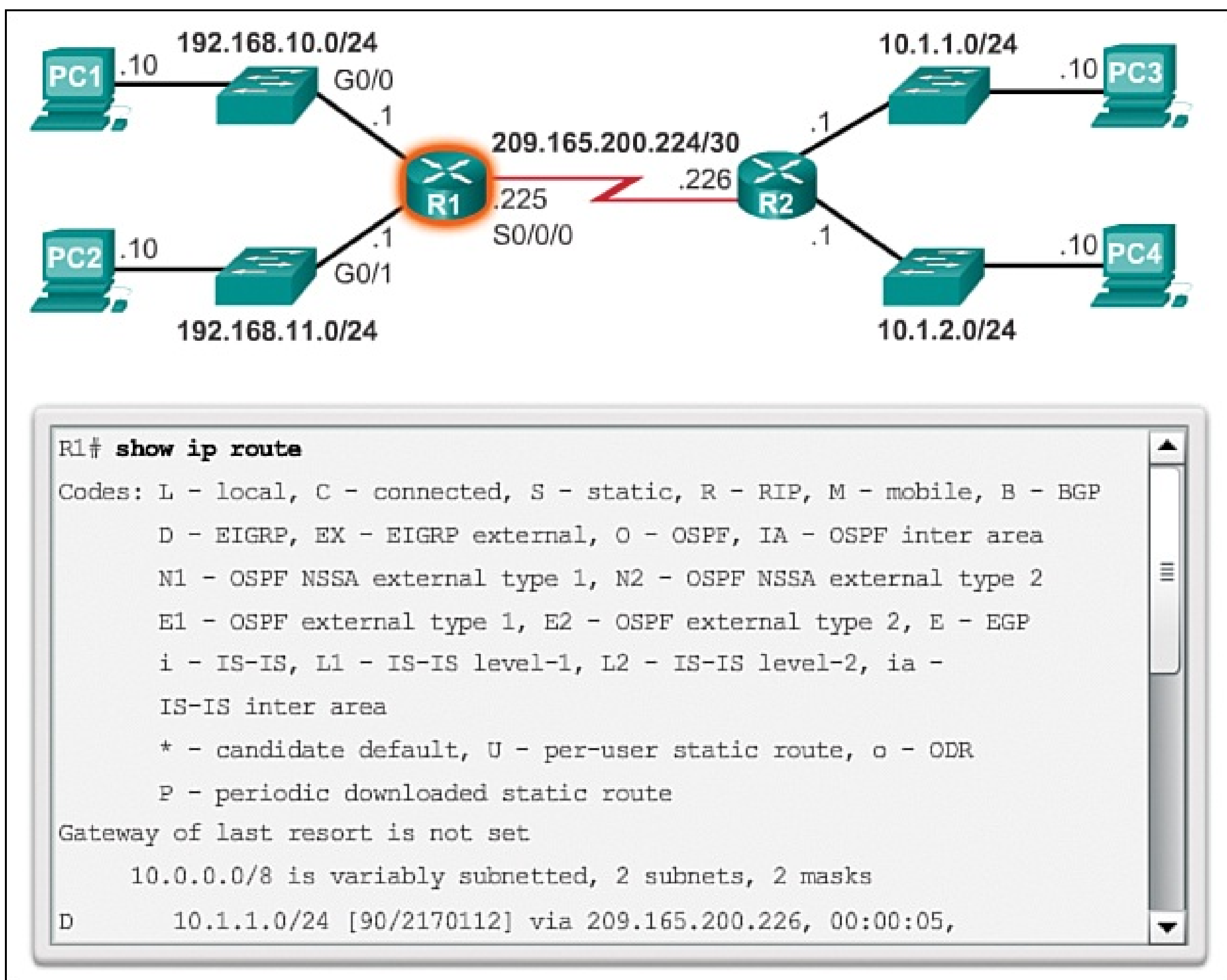
**L**：路由器接口地址。当路由器接收到报文时发送至本地接口而无需转发。

**C**：直连网段。

**O**：通过OSPF从另一个路由器动态学习到的网络。

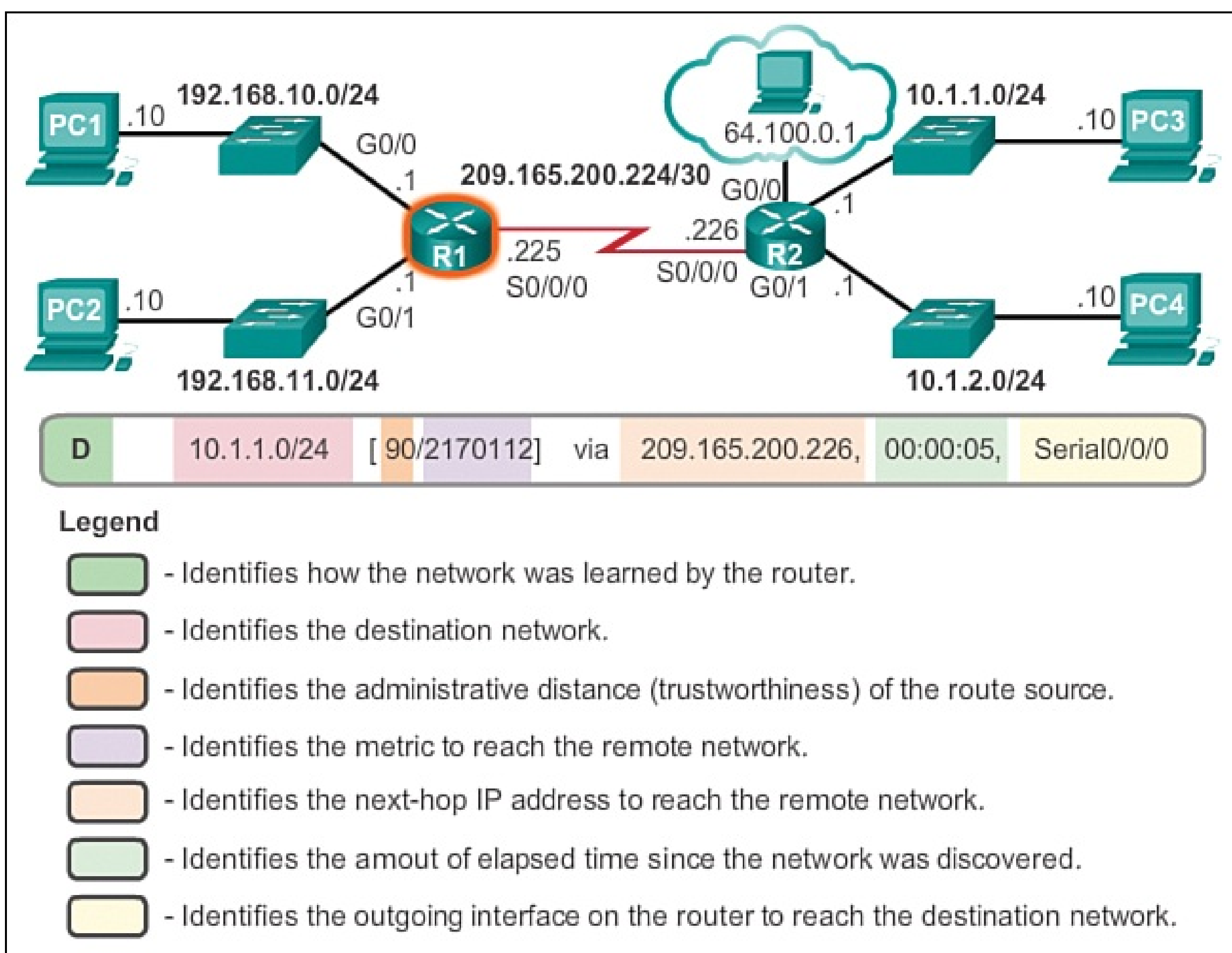
**D**：通过EIGRP从另一个路由器动态学习到的网络。

下图显示了R1的路由表：




远端网络路由表项：

下图显示了R1到远端网络10.1.1.0的表项：




Route source：路径是怎样学习到的。

Destination network：远端网络地址。

Administrative distance：路由来源的可信度。较低值表明优先选择。

Metric：是路由算法用以确定到达目的地的最佳路径的计量标准。较低值表明优先选择。

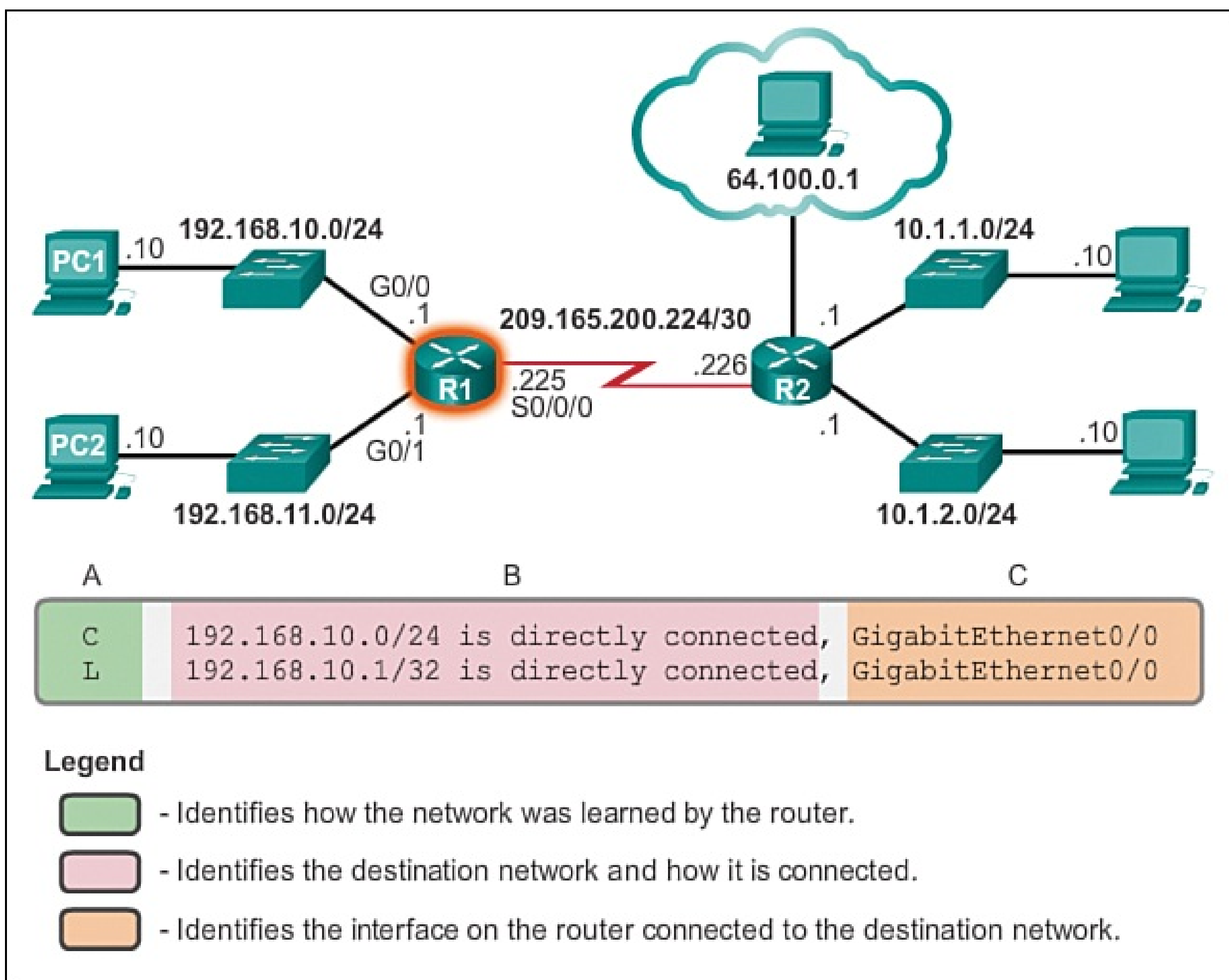
Next hop：转发报文的下一个路由器的IP地址。

Route timestamp：自学习到路径以来过了多少时间。

Outgoing interface：用以转发报文的输出接口。

直连路由表项：

下图显示了R1到直连网络192.168.10.0的路由表项：




在一个接口状态为up/up并添加到IPv4路由表之前，接口必须：

- 指定有效的IPv4或IPv6地址。
- 通过no shutdown命令激活。
- 从另一设备（路由器，交换机，主机等）接收到载体信号。

当接口up之后，该接口的网络作为直连网络添加到路由表中。

# 网络基本功（五）：细说路由（下）

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese



## 介绍

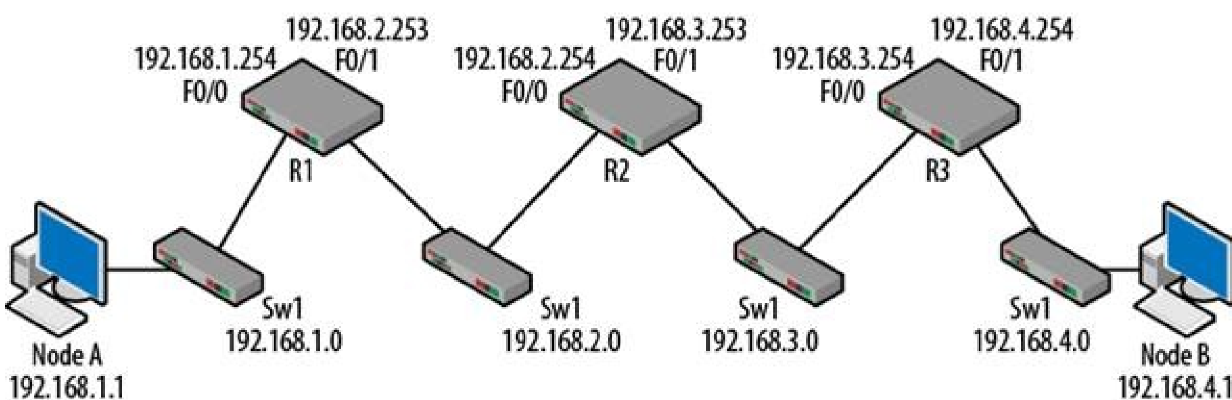
静态路由是指由网络管理员手工配置的路由信息。当网络的拓扑结构或链路的状态发生变化时，网络管理员需要手工去修改路由表中相关的静态路由信息。动态路由是指路由器之间相互通信，传递路由信息，利用收到的路由信息更新路由表的过程。是基于某种协议来实现的。本文详细阐述这两者的实现过程。

## 更多信息

静态路由:

静态路由是指由网络管理员手动配置在路由器上的表项。对于特定的目标地址，以及在小型或稳定的网络环境，手动配置静态路由可以非常成功地应用。通过使用静态路由，网络管理员确定了通向一目标网络的路径。

一个重要的概念是：路由的核心在于下一跳。下一跳是一个特定路由器的角度来看，距离目标地址更近一步的路由器。下图显示了一个中型路由拓扑。从R1的角度来看，R2同时是到达192.168.3.0以及192.168.4.0的下一跳。




初始状态下，除了已经启动的接口和给定的IP地址以外，什么都没有配置。路由器的路由表只会包含直连路由。每一个路由器只知道它接口相连的两个网络。下表显示了这一时刻的路由表。

| R1 | R2 | R3 |
| --- | --- | --- |
| C 192.168.1.0 F0/0 | C 192.168.2.0 F0/0 | C 192.168.3.0 F0/0 |
| C 192.168.2.0 F0/1 | C 192.168.3.0 F0/1 | C 192.168.4.0 F0/1 |

从上表可以看出，路由器不知道整个网络的情况。例如，Node A连接到Switch 1尝试访问Switch 4的Node B。经过主机路由表处理后，A将数据转发至R1的默认网关（192.168.1.254），R1查询自己的路由表并发现没有目标网络的相关信息。于是R1发送ICMP destination unreachable消息。

这个问题怎么解决呢？像这样的小型网络，网络管理员可以在路由器输入路由命令，配置额外的转发信息：

```
ip route destination-network destination-network-mask next-hop-IP-address (forwarding rou
```

例如，以下命令告知R1怎样到达192.168.3.0以及192.168.4.0：

```
ip route 192.168.3.0 255.255.255.0 192.168.2.254
ip route 192.168.4.0 255.255.255.0 192.168.2.254
```

R1上输入命令之后，路由表更新如下所示：

| R1 | R2 | R3 |
| --- | --- | --- |
| C 192.168.1.0 F0/0 | C 192.168.2.0 F0/0 | C 192.168.3.0 F0/0 |
| C 192.168.2.0 F0/1 | C 192.168.3.0 F0/1 | C 192.168.4.0 F0/1 |
| S 192.168.3.0 via 192.168.2.254 | | |
| S 192.168.4.0 via 192.168.2.254 | | |

现在R1理解到达这些网络需要经过R2，但是R2接下来怎么办呢?由于192.168.3.0直接连接到R2，R2可以直接ARP主机。但对于192.168.4.0，R2需要管理员以下命令来协助：

```
ip route 192.168.4.0 255.255.255.0 192.168.3.254
```

路由表相应更新：

| R1 | R2 | R3 |
|---|---|---|
| C 192.168.1.0 F0/0 | C 192.168.2.0 F0/0 | C 192.168.3.0 F0/0 |
| C 192.168.2.0 F0/1 | C 192.168.3.0 F0/1 | C 192.168.4.0 F0/1 |
| S 192.168.3.0 via 192.168.2.254 | S 192.168.4.0 via 192.168.3.254 | |
| S 192.168.4.0 via 192.168.2.254 | | |

目前只成功了一半，报文需要返回。查看R3的路由表，发现路由器不知道怎么找到192.168.1.0。Node A的报文到达之后，Node B尝试回复，但是会从R3收到ICMP destination unreachable的消息。在Node A看来，好像传输从未收到回复。要完成这一过程，需要在所有路由器上对于所有未知网络输入ip route命令来更新路由表。

| R1 | R2 | R3 |
|---|---|---|
| C 192.168.1.0 F0/0 | C 192.168.2.0 F0/0 | C 192.168.3.0 F0/0 |
| C 192.168.2.0 F0/1 | C 192.168.3.0 F0/1 | C 192.168.4.0 F0/1 |
| S 192.168.3.0 via 192.168.2.254 | S 192.168.1.0 via 192.168.2.253 | S 192.168.1.0 via 192.168.3.253 |
| S 192.168.4.0 via 192.168.2.254 | S 192.168.4.0 via 192.168.3.254 | S 192.168.2.0 via 192.168.3.253 |

R2真正的路由表以及在R2上输入的ip route命令如下图所示：

```
Router(config)#
Router(config)#ip route 192.168.1.0 255.255.255.0 192.168.2.253
Router(config)#ip route 192.168.4.0 255.255.255.0 192.168.3.254
Router(config)#
```

```
Gateway of last resort is not set

S    192.168.4.0/24 [1/0] via 192.168.3.254
S    192.168.1.0/24 [1/0] via 192.168.2.253
C    192.168.2.0/24 is directly connected, FastEthernet0/0
C    192.168.3.0/24 is directly connected, FastEthernet0/1
Router#
```

动态路由:

路由协议允许路由器动态共享远端网络的信息以及自动将这信息添加到自己的路由表中。动态路由协议的一大好处在于当拓扑变更时，路由器会交换路由信息，从而能够自动学习新增网络，并且在链路故障时，找到替换路径。

路由协议完成这一功能的方式取决于它所使用的算法以及此协议的操作特性。通常来说，动态路由协议的执行过程如下：

1. 路由器在端口发送和接收路由消息。

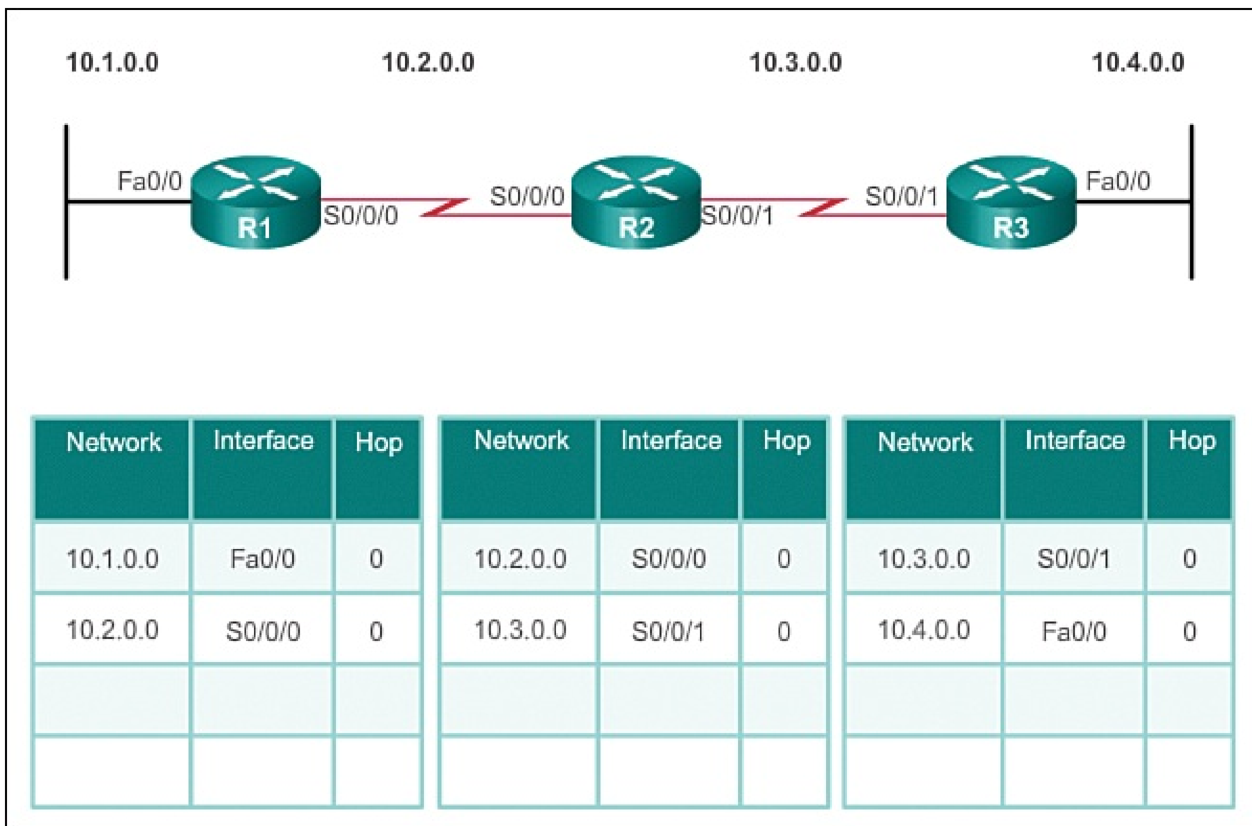
2. 路由器与其他使用相同路由协议的路由器共享路由信息。

3. 路由器交换路由信息来学习远端网络。

4. 当路由器检测到拓扑变化时，路由协议将这一变化通知其他路由器。

网络发现

例如，R1，R2，R3之间的拓扑：




| Network | Interface | Hop |
|---|---|---|
| 10.1.0.0 | Fa0/0 | 0 |
| 10.2.0.0 | S0/0/0 | 0 |
| | | |
| | | |

| Network | Interface | Hop |
|---|---|---|
| 10.2.0.0 | S0/0/0 | 0 |
| 10.3.0.0 | S0/0/1 | 0 |
| | | |
| | | |

| Network | Interface | Hop |
|---|---|---|
| 10.3.0.0 | S0/0/1 | 0 |
| 10.4.0.0 | Fa0/0 | 0 |
| | | |
| | | |

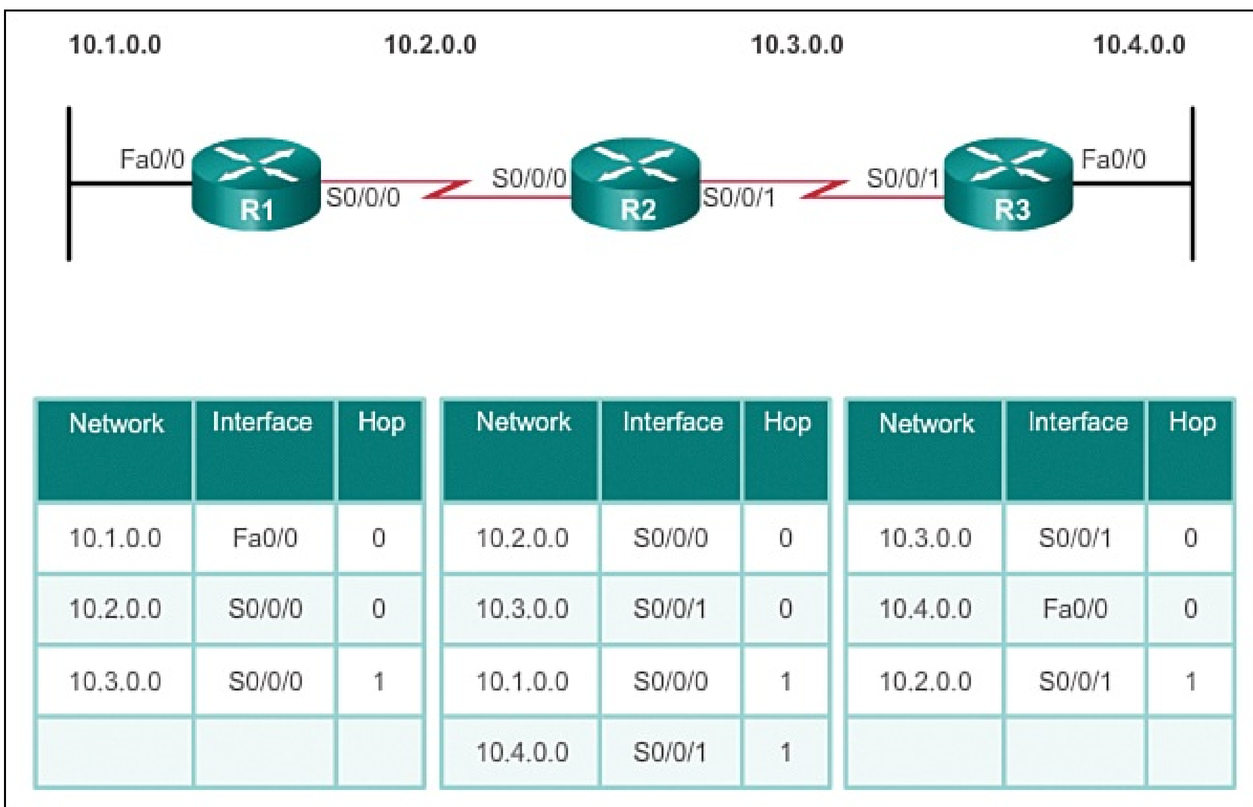
R1：发送10.1.0.0以及10.2.0.0的更新；从R2接收10.3.0.0的信息，跳数加1；在路由表中存储10.3.0.0的信息，metric设为1。

R2：发送10.3.0.0以及10.2.0.0的更新；从R1接收10.1.0.0的信息，跳数加1；在路由表中存储10.1.0.0的信息，metric设为1。从R3接收10.4.0.0的信息，跳数加1；在路由表中存储10.4.0.0的信息，metric设为1。

R3：发送10.3.0.0以及10.4.0.0的更新；从R2接收10.2.0.0的信息，跳数加1；在路由表中存储10.2.0.0的信息，metric设为1。

交换路由信息

路由器周期性的更新信息。在最初的网络发现结束后，每个路由器通过发送和接收以下更新来继续收敛的过程：




| Network | Interface | Hop |
|---|---|---|
| 10.1.0.0 | Fa0/0 | 0 |
| 10.2.0.0 | S0/0/0 | 0 |
| 10.3.0.0 | S0/0/0 | 1 |
| | | |

| Network | Interface | Hop |
|---|---|---|
| 10.2.0.0 | S0/0/0 | 0 |
| 10.3.0.0 | S0/0/1 | 0 |
| 10.1.0.0 | S0/0/0 | 1 |
| 10.4.0.0 | S0/0/1 | 1 |

| Network | Interface | Hop |
|---|---|---|
| 10.3.0.0 | S0/0/1 | 0 |
| 10.4.0.0 | Fa0/0 | 0 |
| 10.2.0.0 | S0/0/1 | 1 |
| | | |

R1：发送10.1.0.0，10.2.0.0以及10.3.0.0的更新；从R2接收10.4.0.0的信息，跳数加1；在路由表中存储10.4.0.0的信息，metric设为2；从R2收到相同的10.3.0.0的更新，metric为1，不作更新。

R2：发送10.1.0.0，10.2.0.0，10.3.0.0以及10.4.0.0的更新；从R1接收10.1.0.0的信息，不作更新；从R3接收10.4.0.0的信息，不作更新。

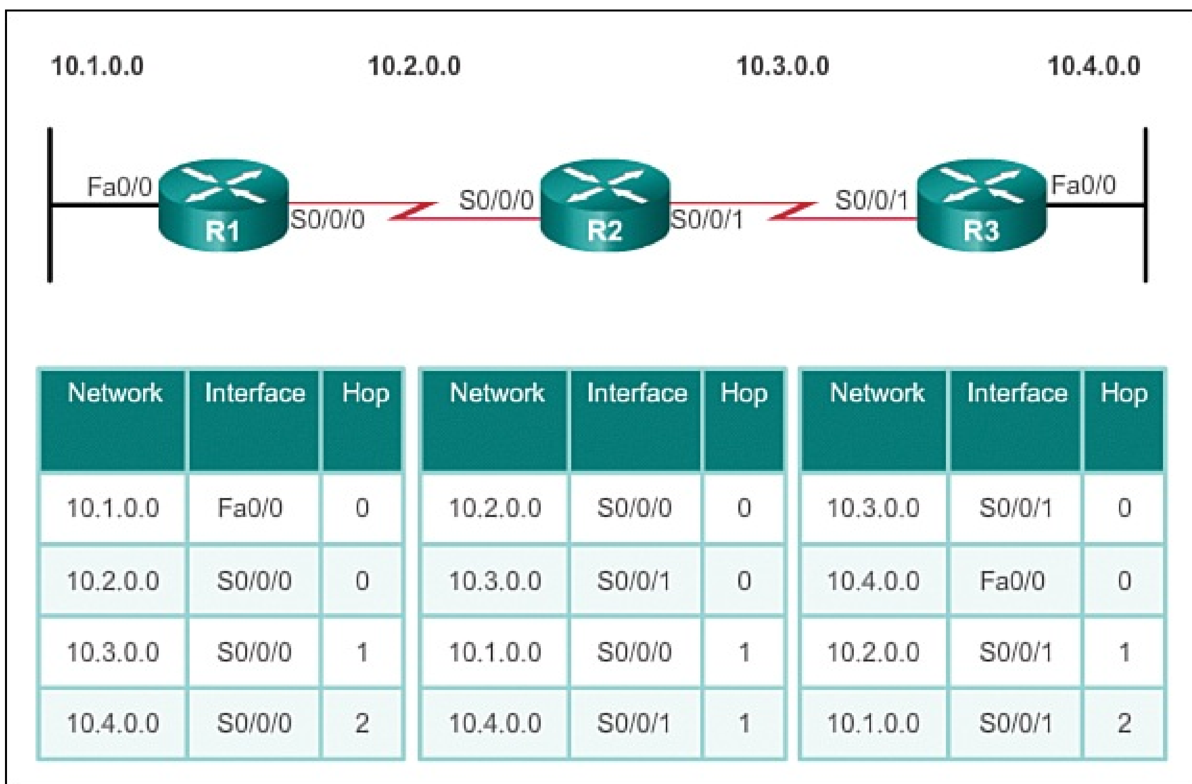
R3：发送10.2.0.0，10.3.0.0以及10.4.0.0的更新；从R2接收10.1.0.0的信息，跳数加1；在路由表中存储10.1.0.0的信息，metric设为2；从R2收到相同的10.2.0.0的更新，metric为1，不作更新。

距离矢量路由协议切断了邻居路由之间的环路，也称为水平分割。水平分割阻止信息从同一端口接收之后再发送出去。例如，R2不会从Serial 0/0/0端口发送网络10.1.0.0的信息，因为R2从Serial 0/0/0学习了10.1.0.0。

网络中的路由器收敛了信息之后，路由器可以使用路由表来决定到达目的地的最佳路径。不同的路由协议有不同的计算最佳路径的方法。

路由收敛

当所有路由器对于整个网络有准确的更新之后，达到路由收敛状态，如下图所示：




| Network | Interface | Hop | Network | Interface | Hop | Network | Interface | Hop |
|---|---|---|---|---|---|---|---|---|
| 10.1.0.0 | Fa0/0 | 0 | 10.2.0.0 | S0/0/0 | 0 | 10.3.0.0 | S0/0/1 | 0 |
| 10.2.0.0 | S0/0/0 | 0 | 10.3.0.0 | S0/0/1 | 0 | 10.4.0.0 | Fa0/0 | 0 |
| 10.3.0.0 | S0/0/0 | 1 | 10.1.0.0 | S0/0/0 | 1 | 10.2.0.0 | S0/0/1 | 1 |
| 10.4.0.0 | S0/0/0 | 2 | 10.4.0.0 | S0/0/1 | 1 | 10.1.0.0 | S0/0/1 | 2 |

收敛时间是路由器分享信息，计算最佳路径，更新路由表的时间。收敛同时是协作并且独立的。路由器相互之间共享信息但是必须各自独立的计算自己路由拓扑改变所带来的影响。 由于它们各自独立地关于新的拓扑达成一致，于是说它们收敛于这种一致。

# 网络基本功（六）：链路聚合

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

链路聚合是在两个设备间使用多个物理链路创建一个逻辑链路的功能。这种方式允许物理链路间共享负载。交换机网络中使用的一种链路聚合的方法是EtherChannel。EtherChannel可以通过思科的端口聚合协议（Port Aggregation Protocol, PAgP）或链路聚合协议（Link Aggregation Protocol, LACP）来配置或协商。

## 更多信息

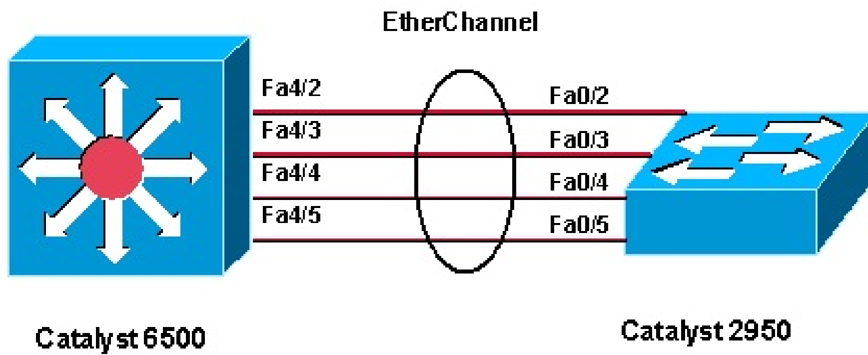
**EtherChannel：**

EtherChannel本来是由思科开发，将若干Fast Ethernet或Gigabit Ethernet捆绑成一个逻辑通道的交换机到交换机的LAN连接技术。配置了EtherChannel之后的虚拟接口称为一个port channel。物理接口捆绑在一起，成为一个port channel interface。思科最早称之为EtherChannel Fast EtherChannel(FEC)，也称为Gigabit EtherChannel(GEC)，非思科公司常将链路聚合简写为LAG。

通过EtherChannel，一个逻辑链路的速度等于所有物理链路的总和。例如，如果你用4个100 Mbps的以太网链路创建1个EtherChannel，则EtherChannel的速度是400 Mbps。但是也会有一些问题，并不是在所有情况下增加的容量都确实等于物理链路的速度之和。例如，四个1 Gbps链路组成的EtherChannel，默认每一个会话的速度还是限制在1 Gbps。

默认情况下EtherChannel按照报文的目的MAC地址，给它指定一个物理链接。这也意味着EtherChannel上一个工作站与另一个服务器通信，只会使用到一条物理链路。实际上，EtherChannel上所有目的地为该服务器的数据流都只会走这一条物理链路。也就是说，一个用户同一时刻只会得到1 Gbps。这种模式也可以更改为每一个报文在不同的物理链路上发送，当有多个不同的目的地址时，每一条路径都可以得到利用。

EtherChannel创建的是一对一的关系，即一个EtherChannel连接两个设备。可在两台交换机之间，或在一个激活了EtherChannel的服务器和一台交换机之间创建一个EtherChannel连接。但是，同一个EtherChannel连接无法将数据流发送到两台交换机。




**EtherChannel负载均衡：**

如前所述，EtherChannel默认情况下并不真的为各链路速度之和，只是在特定的链路发送特定的报文，给人的感知速度为所有链路的速度总和。EtherChannel 帧分发使用 Cisco 专有的hash算法。 该算法是确定性算法； 如果使用相同的地址和会话信息，则总是散列到通道中的同一端口。 此方法可避免无序传送数据包。这一算法中很重要的一点是，并不保证物理链路之间完全地均衡。

该算法将目的MAC地址值hash成0-7的范围。无论EtherChannel中有多少链路都是同样的值。每一条物理链路都指定这八个值中的一个或多个，取决于EtherChannel中共有几条链路。

下图显示了按照这种算法报文是怎样分布的，注意到分布并不总是均衡的。

| Link number \ Number of physical links | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 |
| 2 | 1 | 1 | 2 | 2 | 2 | 3 | 4 |
| 3 | 1 | 1 | 1 | 2 | 2 | 2 | |
| 4 | 1 | 1 | 1 | 1 | 2 | | |
| 5 | 1 | 1 | 1 | 1 | | | |
| 6 | 1 | 1 | 1 | | | | |
| 7 | 1 | 1 | | | | | |
| 8 | 1 | | | | | | |

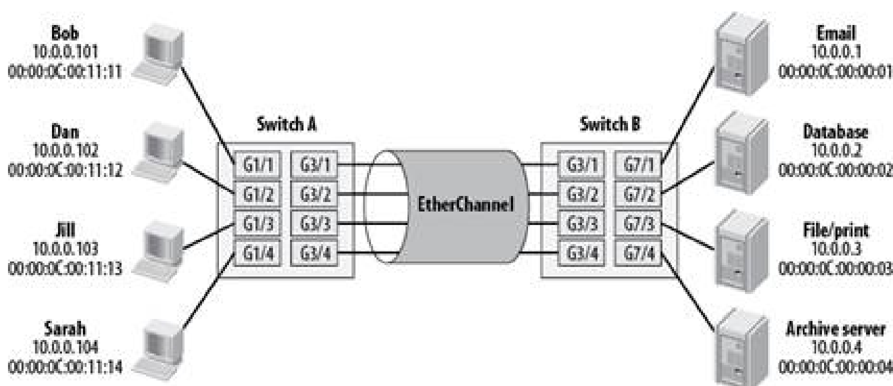
有八条物理链路的EtherChannel，每条链路指定单一值。有六条链路的EtherChannel，两条链路指定两个值，剩下四条链路指定四个值。这意味着两条链路（理论上均衡分布）会收到比剩余四条链路多一倍的数据流。从这张图很明显的看出，要使流量在各链路间均衡的分布（理想情况下），应当设置1，2，4，或8条物理链路。无论决定链路的信息是什么，算法都会将链路值hash为0-7。

用户可根据需求对算法进行更改。默认行为是使用目的MAC地址，但是，按照软硬件版本的不同，还可以有如下选项：

- 源MAC地址
- 目的MAC地址
- 源和目的MAC地址
- 源IP地址
- 目的IP地址
- 源和目的IP地址
- 源端口
- 目的端口
- 源和目的端口

更改默认设置的原因由应用情况而定。下图显示了一种相对普遍的布局：

一组用户连接到交换机A，通过EtherChannel连接到交换机B。默认按照每一个报文的目的MAC地址做负载均衡。但是，比较常见的情况是一台服务器的流量显著高于其他服务器。

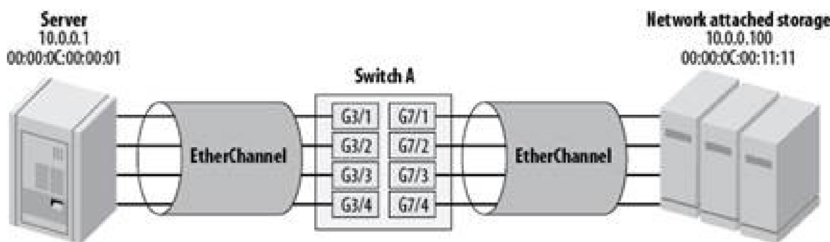



让我们假设该网络中email服务器接收到多于1 Gbps流量，而其他服务器大约为50Mbps。使用基于目的MAC地址的方法会导致在EtherChannel丢包，因为目的地为email 服务器 MAC地址的报文会走同一条物理链路。一条链路发生过载时报文不会分散到其他链路，只会丢弃。在这种一台服务器接收流量超大的情况下，目的MAC地址负载均衡就不合理了。而根据源MAC地址负载均衡更为合适。

另一点需要记住的是，负载均衡算法只适用于EtherChannel上发送的报文。它并没有双向功能。在交换机A上使用基于源MAC地址的算法可能比较合适，但对于交换机B不一定合适，因为email服务器是使用最多的服务器。当报文从email服务器返回，源MAC地址就是它自己本身。因此，如果我们在交换机B上使用基于源MAC地址的负载均衡算法，就会碰到一开始同样的问题。

这种情况下，解决方法是在交换机A使用基于源MAC地址的负载均衡算法，而在交换机B使用目的MAC地址的算法。如果所有服务器在一台交换机而所有用户在另一台，这一解决方案是有效的。但现实中更常见的情况是所有这些设备都连接在一台交换机上，这时就没那么走运了。

下图显示了一个比较有趣的问题。一台服务器通过EtherChannel连接到交换机A，一台NAS也通过EtherChannel连接到交换机A。服务器的所有文件系统都挂在到NAS设备上，服务器作为一台服务超过5000人的数据库服务器负载很大。服务器和NAS之间的带宽需求超过2Gbps。




目前没有解决这一问题的简单的方法。不能使用源MAC地址或目的MAC地址做负载均衡，因为每种情况都只有一个地址。同样的理由，也不能用源和目的MAC地址结合，源和目的IP地址结合的方法。也不能基于源或目的端口号，因为一旦协商结束后，它们就不会改变。一种可能的方法是，驱动支持的情况下，改变服务器和/或NAS设备，每一个link都有自己的MAC地址，但是报文还是会从其中一个地址发出另一个地址接收。

唯一的解决方法是手动负载均衡或采用更快的链接。将链路分为4个1Gbps，每一个有自己的IP网络，每个连接mount挂载不同的文件系统可以解决这一问题。有点太过复杂的话，直接使用更快的物理连接，如10 Gbps。

# 网络基本功（七）：细说IP地址与子网

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese

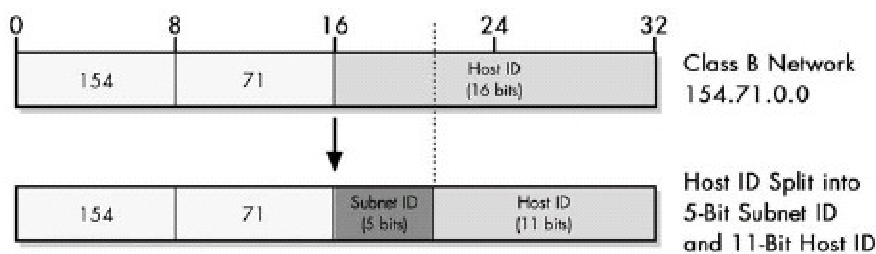



## 介绍

起初，IP地址只有两层结构：网络与主机。子网地址向其中添加了一层新的结构：不同于仅有主机，网络有分为子网与主机。每一个子网的功能近乎于完整的网络。子网的添加构成了三层网络结构：包含子网的网络，各自由若干主机构成。IP地址由此被分为三个部分：网络ID，子网ID与主机ID。IP地址长度仍固定为32位，其中，A类网络8位子网掩码，B类网络16位子网掩码，C类网络24位子网掩码。

## 更多信息

对于每一类网络，网络数以及每一网络中包含的主机数，决定了它们各自占用多少比特位。这一准则同样适用于如何划分子网与主机。子网数量为2的子网ID次方，每一子网内的主机数为2的主机ID次方。假设一个B类网络154.71.0.0，网络ID占16位（154.71），主机ID占16位。没有子网的情况下一共可容纳65，534台主机。按照实际需求将16位划分为子网与主机：1位子网16位主机，或2与14，3与13。。。如下图所示，划分为5位子网与11位主机，子网数越多，主机数越少。




搭建IP子网时，如何划分子网与主机数是最重要的问题之一。子网所占位取决于整个网络中的物理子网数，每一子网中的主机数不能超过子网划分所允许的最大数量。

**IP子网掩码，表示法以及子网计算:**

在没有子网的网络环境下，路由器通过IP地址的前八位来决定是哪一类型的网络，从而它们知道哪些是网络ID哪些是主机ID。划分子网时，路由器也需要知道主机ID是如何划分成子网ID与主机ID的，但是划分方法可以是任意组合，也没有办法从IP地址看出来。因此，必须有额外的信息告知解析IP地址的设备，这一信息称为子网掩码，以32比特数的形式呈现。

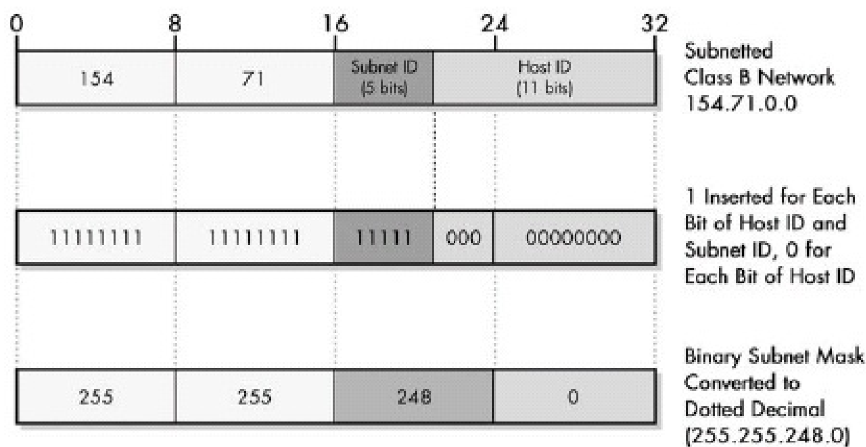
掩码位的1和0结合布尔函数与和或的功能对于地址中的比特位进行选择或清除。子网掩码中的32位对应于IP地址相同位置上的数字。掩码位为1时，则地址中该位作为网络ID或子网ID，而掩码位为0时，则地址中该位表示主机ID。

子网掩码为**1**：将IP地址中的0或1与1进行与操作，即：当子网掩码位为1，IP地址保持不变。

子网掩码为**0**：任何数和0做与操作都是0，即：当子网掩码位为0，IP地址清零。

因此，将子网掩码应用于IP地址，网络ID和子网ID保持不变，移除主机ID。执行此功能的路由器由此获得子网地址，因为它知道网络类型，因此能够区分网络位与子网地址位。

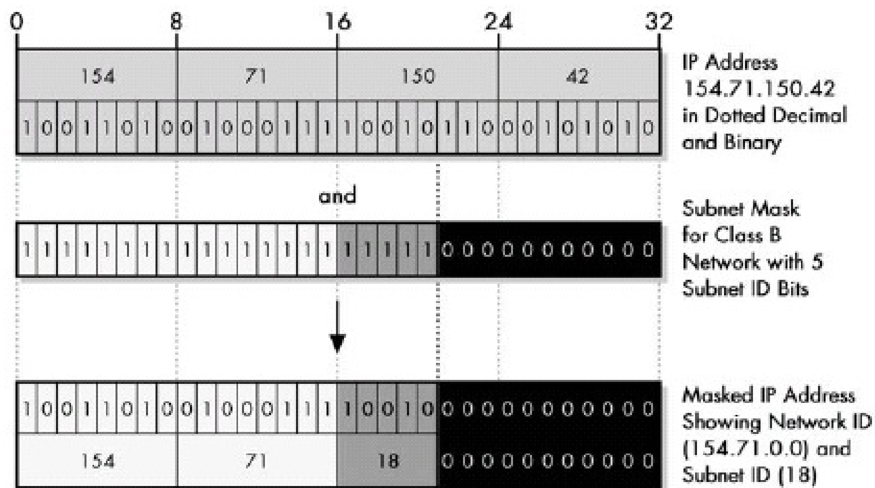
举例来说，假设将B类网络154.71.0.0划分5位为子网ID，11位为主机ID。因此，子网掩码有16个1代表网络部分（B类网络），接下来5个1作为子网部分，11个0用作主机ID。二进制数表示为11111111 11111111 **11111**000 00000000，十进制数表示为255.255.248.0。




举例：

假设有一台主机IP地址154.71.150.42，路由器需要找出该主机位于哪一子网，则它的掩码操作如下图所示：

| Component | Octet 1 | Octet 2 | Octet 3 | Octet 4 |
|---|---|---|---|---|
| IP Address | 10011010 (154) | 01000111 (71) | 10010110 (150) | 00101010 (42) |
| Subnet Mask | 11111111 (255) | 11111111 (255) | **11111**000 (248) | 00000000 (0) |
| Result of AND Masking | 10011010 (154) | 01000111(71) | **10010**000 (144) | 00000000 (0) |




结果，154.71.150.42所属的子网为154.71.144.0。另一台路由器能够从中区分出网络ID与子网ID，因为地址的前两个比特位是10，是一个B类网络。所以网络ID占16位，子网ID一定是17至21。这里，子网是10010，或子网18。

提一个问题：既然子网掩码只是将网络地址划分出网络部分与子网部分，那为什么还要使用另外的32位比特数255.255.248.0，而不直接将IP地址第21位指定为分界线呢？这是有历史原因的：因为需要考虑不连续的掩码情况。同时，它也能够让路由器进行快速的掩码操作来找出子网地址。

除了将16位划分为5位子网ID与11位主机ID，标准也允许前2位用作子网ID，4位用作主机ID，之后3位用作子网ID，7位用作主机ID。因此子网掩码为11000011 10000000。当然，这会造成混淆，是不推荐的，实际中也没有人会这么做。

鉴于非连续掩码实际不会应用，以及现今的计算机速度大幅提升，新的表达法为154.71.150.42/21。

**IP子网掩码设定:**

假设B类网络154.71.0.0，没有子网的话一共有65,534台主机。划分子网时，按照以下方法：

- 1位用作子网ID，15位用作主机ID：那么子网数为2^1，第一个子网是0，第二个子网是1。每一个子网的主机数是2^15-2，或32，766。
- 2位用作子网ID，14位用作主机ID：那么子网数为2^2，四个子网0，1，2，3。每一个子网的主机数是2^14-2，或16，382。

子网与主机ID位的划分取决于子网数与子网中最大主机数。假设一个B类网络中有10个子网，需要4位表示子网（2^4=16，2^3=8），12位用作主机ID，每一子网最多4，094台主机。

如果你有20个子网，每一子网3，000台主机，那么就会碰到问题。需要5位表示20个子网，而3，000台主机需要12位。这时需要重新组织物理网络，如果无法做到，就需要第二个B类网络。

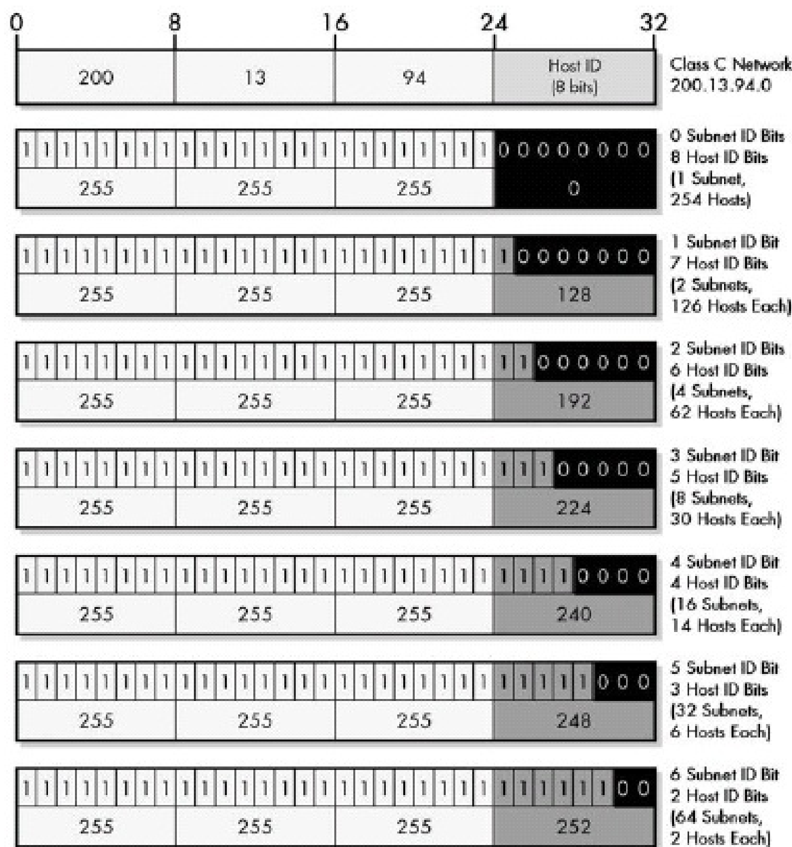
自定义子网掩码的方法是：从指定网络类型的默认子网掩码中，从最左边的0位开始，按照需要的子网数将0改为1。假设C类网络200.13.94.0，最后8位可供划分子网与主机，则有6种不同的划分方法。假如使用3位作为子网ID，5位作为主机ID，那么：

默认C类网络子网掩码：11111111 11111111 11111111 00000000

将最左边的3位0改为1：11111111 11111111 11111111 **111**00000

即子网掩码为：255.255.255.224。

通常情况下，所有子网大小必须相同。因此，最大一个子网的主机数决定了需要多少位比特用作主机ID。因此前例中，前19个子网每个子网最多100台主机，而第20个子网需要3000个主机，就会碰到问题。这种情况下，需要将最后一个过大的子网拆成若干个小的子网。



# 网络基本功（八）：细说TCP滑动窗口

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

将TCP与UDP这样的简单传输协议区分开来的是它传输数据的质量。TCP对于发送数据进行跟踪，这种数据管理需要协议有以下两大关键功能：

可靠性：保证数据确实到达目的地。如果未到达，能够发现并重传。

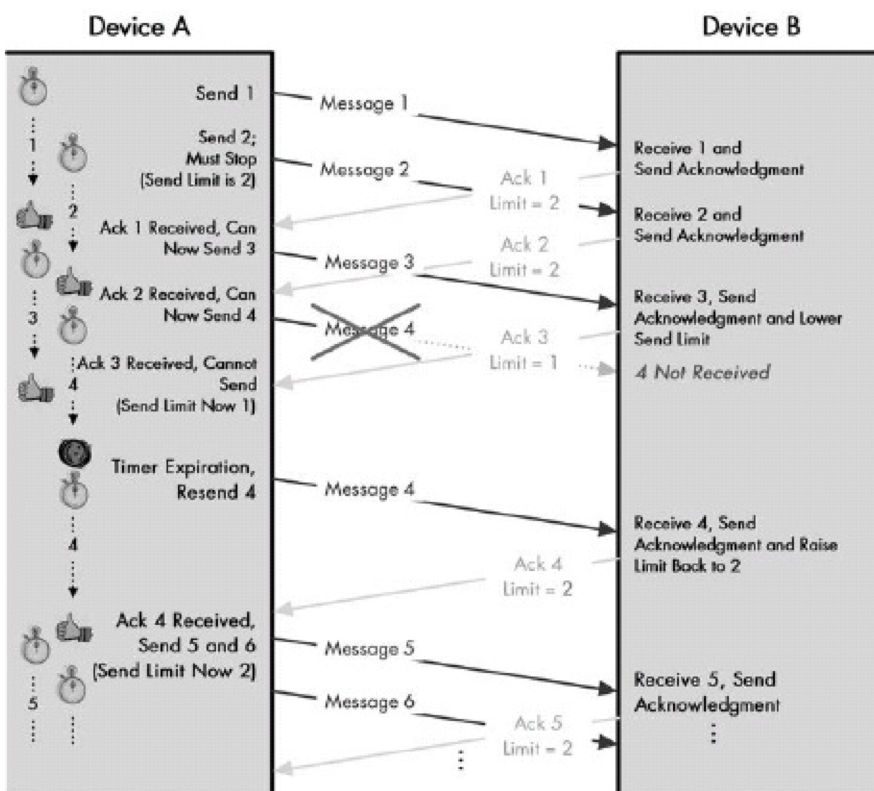
数据流控：管理数据的发送速率，以使接收设备不致于过载。

要完成这些任务，整个协议操作是围绕滑动窗口确认机制来进行的。因此，理解了滑动窗口，也就是理解了TCP。

## 更多信息

**TCP**面向流的滑动窗口确认机制:

TCP将独立的字节数据当作流来处理。一次发送一个字节并接收一次确认显然是不可行的。即使重叠传输（即不等待确认就发送下一个数据），速度也还是会非常缓慢。



TCP消息确认机制如上图所示，首先，每一条消息都有一个识别编号，每一条消息都能够被独立地确认，因此同一时刻可以发送多条信息。设备B定期发送给A一条发送限制参数，制约设备A一次能发送的消息最大数量。设备B可以对该参数进行调整，以控制设备A的数据流。

为了提高速度，TCP并没有按照字节单个发送而是将数据流划分为片段。片段内所有字节都是一起发送和接收的，因此也是一起确认的。确认机制没有采用message ID字段，而是使用的片段内最后一个字节的sequence number。因此一次可以处理不同的字节数，这一数量即为片段内的sequence number。

**TCP**数据流的概念划分类别

假设A和B之间新建立了一条TCP连接。设备A需要传送一长串数据流，但设备B无法一次全部接收，所以它限制设备A每次发送分段指定数量的字节数，直到分段中已发送的字节数得到确认。之后，设备A可以继续发送更多字节。每一个设备都对发送，接收及确认数据进行追踪。

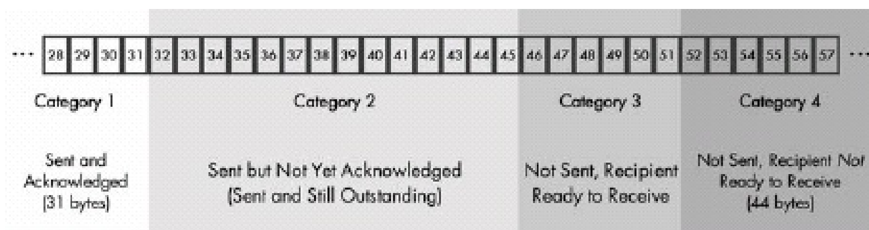
如果我们在任一时间点对于这一过程做一个“快照”，那么我们可以将TCP buffer中的数据分为以下四类，并把它们看作一个时间轴：

**1.** 已发送已确认 数据流中最早的字节已经发送并得到确认。这些数据是站在发送设备的角度来看的。如下图所示，31个字节已经发送并确认。

**2.** 已发送但尚未确认 已发送但尚未得到确认的字节。发送方在确认之前，不认为这些数据已经被处理。下图所示14字节为第2类。

**3.** 未发送而接收方已**Ready** 设备尚未将数据发出，但接收方根据最近一次关于发送方一次要发送多少字节确认自己有足够空间。发送方会立即尝试发送。如图，第3类有6字节。

**4.** 未发送而接收方**Not Ready** 由于接收方not ready，还不允许将这部分数据发出。




接收方采用类似的机制来区分已接收并已确认，尚未接受但准备好接收，以及尚未接收并尚未准备好接收的数据。实际上，收发双方各自维护一套独立的变量，来监控发送和接收的数据流落在哪一类。

**Sequence Number**设定与同步：

发送方和接收方必须就它们将要为数据流中的字节指定的sequence number达成一致。这一过程称为同步，在TCP连接建立时完成。为了简化假设第一个字节sequence number是1，按照上图示例，四类字节如下：

1. 已发送已确认字节1至31。

2. 已发送但尚未确认字节32至45。

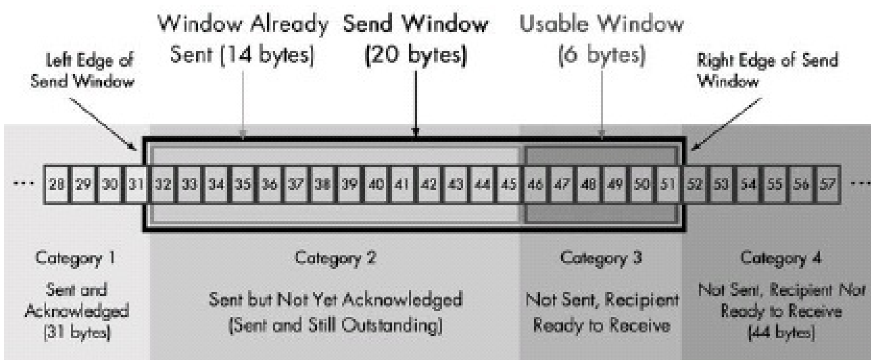
3. 未发送而接收方已Ready字节46至51。

4. 未发送而接收方Not Ready字节52至95。

发送窗口与可用窗口：

整个过程关键的操作在于接收方允许发送方一次能容纳的未确认的字节数。这称为发送窗口，有时也称为窗口。该窗口决定了发送方允许传送的字节数，也是2类和3类的字节数之和。因此，最后两类（接收方准备好而尚未发送，接收方未准备好）的分界线在于添加了从第一个未确认字节开始的窗口。本例中，第一个未确认字节是32，整个窗口大小是20。

可用窗口的定义是：考虑到正在传输的数据量，发送方仍被允许发送的数据量。实际上等于第3类的大小。左边界就是窗口中的第一个字节（字节32），右边界是窗口中最后一个字节（字节51）。概念的详细解释看下图。




可用窗口字节发送后**TCP**类目与窗口大小的改变：

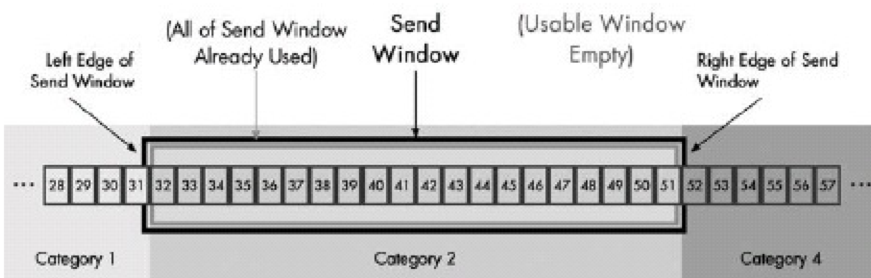
当上图中第三类的6字节立即发送之后，这6字节从第3类转移到第2类。字节变为如下：

1. 已发送已确认字节1至31。

2. 已发送但尚未确认字节32至51。

3. 未发送而接收方已Ready字节为0。

4. 未发送而接收方Not Ready字节52至95。




确认处理以及窗口缩放：

过了一段时间，目标设备向发送方传回确认信息。目标设备不会特别列出它已经确认的字节，因为这会导致效率低下。目标设备会发送自上一次成功接收后的最长字节数。

例如，假设已发送未确认字节（32至45）分为4段传输：32-34，35-36，37-41，42-45。第1，2，4已经到达，而3段没有收到。接收方只会发回32-36的确认信息。接收方会保留42-45但不会确认，因为这会表示接收方已经收到了37-41。这是很必要的，因为TCP的确认机制是累计的，只使用一个数字来确认数据。这一数字是自上一次成功接收后的最长字节数。假设目标设备同样将窗口设为20字节。

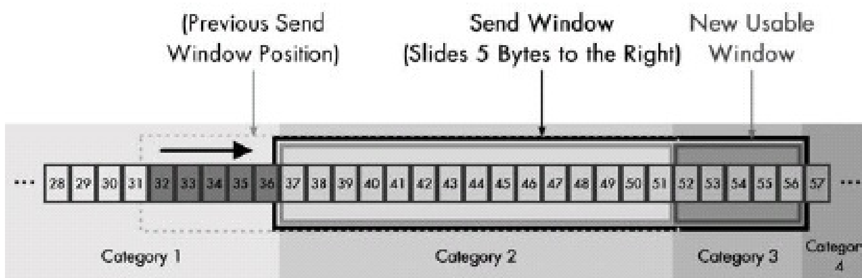
当发送设备接收到确认信息，则会将一部分第2类字节转移到第1类，因为它们已经得到了确认。由于5个字节已被确认，窗口大小没有改变，允许发送方多发5个字节。结果，窗口向右滑动5个字节。同时5个字节从第二类移动到第1类，5个字节从第4类移动至第3类，为接下来的传输创建了新的可用窗口。因此，在接收到确认信息以后，看起来如下图所示。字节变为如下：

1. 已发送已确认字节1至36。

2. 已发送但尚未确认字节37至51。

3. 未发送而接收方已Ready字节为52至56。

4. 未发送而接收方Not Ready字节57至95。




每一次确认接收以后，这一过程都会发生，从而让窗口滑动过整个数据流以供传输。

处理丢失确认信息：

但是丢失的42-45如何处理呢？在接收到第3段（37-41）之前，接收设备不会发送确认信息，也不会发送这一段之后字节的确认信息。发送设备可以将新的字节添加到第3类之后，即52-56。发送设备之后会停止发送，窗口停留在37-41。

TCP包括一个传输及重传的计时机制。TCP会重传丢失的片段。但有一个缺陷是：因为它不会对每一个片段分别进行确认，这可能会导致其他实际上已经接收到的片段被重传（比如42至45）。

# 网络基本功（九）：细说TCP重传

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

TCP的主要任务是很简单：打包和发送数据。TCP与其他协议的不同之处在于使用滑动窗口来管理基本数据收发过程，同时确保数据流的有效及可靠传输，从而不致发送速率明显快于接收速率。本文将描述TCP是如何确保设备可靠、有效地进行传输的。首先阐述TCP检测丢失片段以及重传的基本方法，之后介绍TCP如何判断一个片段为丢失片段。

## 更多信息

**TCP**片段重传计时器以及重传队列**:**

检测丢失片段并对之重传的方法概念上是很简单的。每一次发送一个片段，就开启一个重传计时器。计时器有一个初始值并随时间递减。如果在片段接收到确认之前计时器超时，就重传片段。TCP使用了这一基本技术，但实现方式稍有不同。原因在于为了提高效率需要一次处理多个未被确认的片段，以保证每一个在恰当的时间重传。TCP按照以下特定顺序工作：

放置于重传队列中，计时器开始 包含数据的片段一经发送，片段的一份复制就放在名为重传队列的数据结构中，此时启动重传计时器。因此，在某些时间点，每一个片段都会放在队列里。队列按照重传计时器的剩余时间来排列，因此TCP软件可追踪那几个计时器在最短时间内超时。

确认处理 如果在计时器超时之前收到了确认信息，则该片段从重传队列中移除。

重传超时 如果在计时器超时之前没有收到确认信息，则发生重传超时，片段自动重传。当然，相比于原片段，对于重传片段并没有更多的保障机制。因此，重传之后该片段还是保留在重传队列里。重传计时器被重启，重新开始倒计时。如果重传之后没有收到确认，则片段会再次重传并重复这一过程。在某些情况下重传也会失败。我们不想要TCP永远重传下去，因此TCP只会重传一定数量的次数，并判断出现故障终止连接。

但是我们怎样知道一个片段被完全确认呢？重传是基于片段的，而TCP确认信息是基于序列号累积的。每次当设备A发送片段给设备B，设备B查看该片段的确认号字段。所有低于该字段的序列号都已经被设备A接收了。因此，当片段中所发送的所有字节的序列号都比设备A到设备B的最后一个确认号小的时候，一个从设备B发到设备A的片段被认为是确认了。这是通过计算片段中最后一个序列号结合片段的数据字段来实现的。

让我们以下图为例来说明一下确认和重传是怎样工作的。假设连接中的服务器发出了四个连续片段（号码从1开始）

片段**1** 序列号字段是1片段长度80。所以片段1中最后一个序列号是80。

片段**2** 序列号是81片段长度是120。片段2中最后一个序列号是200。

片段**3** 序列号是201片段长度是160。片段3中最后一个序列号是360。

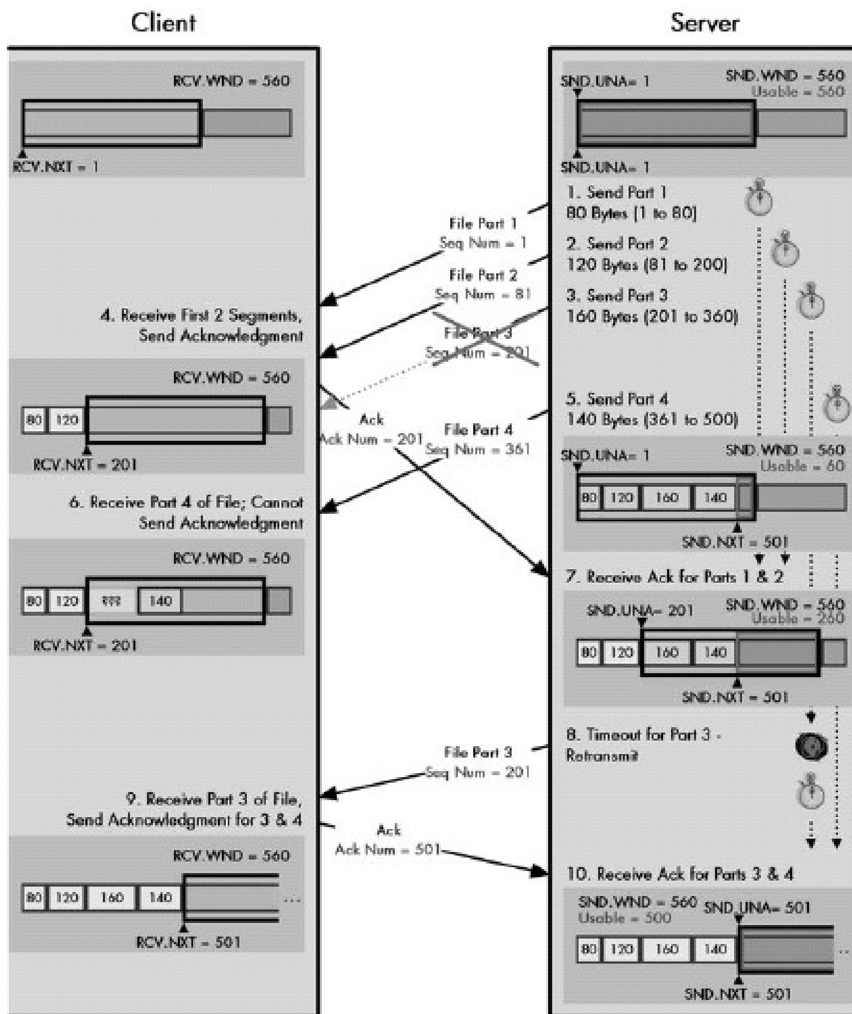
片段**4** 序列号是361片段长度是140。片段3中最后一个序列号是500。

这些片段是一个接一个发送的，而无需等待前一个发送得到确认。这是TCP滑动窗口的一个主要优势（细说TCP滑动窗口）。

假设客户端接收到前两个传输，它会发回一条确认消息确认号为201。从而告知服务器前两个片段已经被客户端成功接收了，它们从重传队列中移除（并且服务器发送窗口右移200字节）。在接收到确认号361或更高的片段之前，片段3会保留在重传队列中；片段4需要确认号501或更高。

现在，让我们进一步假设传输过程中片段3丢失了，但片段4被接收到了。客户端将片段4保存在接收buffer中，但是不需要确认，因为TCP是累积确认机制——确认片段4表示片段3也接收到了，但实际上并没有。因此，客户端需要等待片段3。实际上，服务器端片段3的重传计时器会超时，服务器之后重传片段3。之后客户端收到，然后发送片段3和4的确认信息给服务器。

还有一个重要的问题，服务器将如何处理片段4呢？虽然客户端在等待片段3，服务器没有收到反馈，所以它并不知道片段3丢失了，同样它也不知道片段4发生了什么（以及接下来传输的数据）。很有可能客户端已经接收到了片段4但是不能确认，也有可能片段4也丢失了。一些实现中会选择仅仅重传片段3，也有些会把3和4都重传。




最后一个问题是重传队列中所使用片段重传计时器的值。如果设置过低，会发生过量重传，如果设置过高，重传丢失片段会减弱性能。必须通过一个称为自适应重传的过程来动态调整这个值，接下来的章节会讲到。

# 网络基本功（十）：细说TCP确认机制

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese



## 介绍

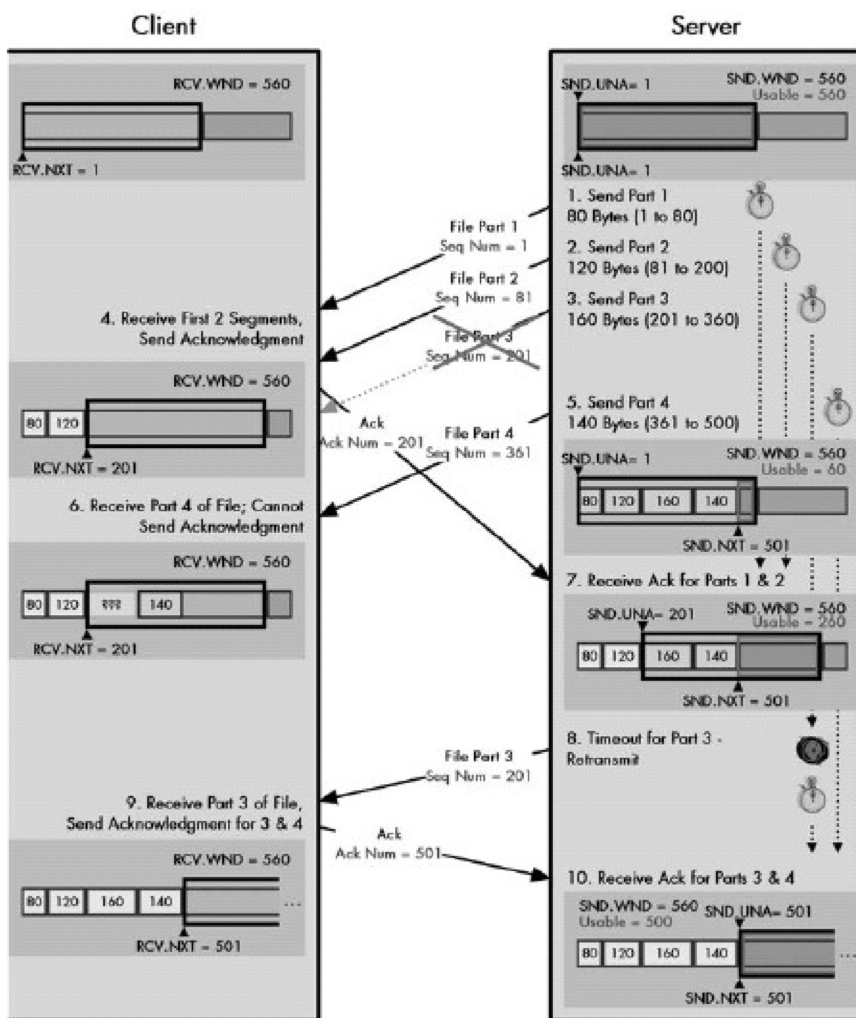
在TCP确认机制中，无法有效处理非连续TCP片段。确认号表明所有低于该编号的sequence number已经被发送该编号的设备接收。如果我们收到的字节数落在两个非连续的范围内，则无法只通过一个编号来确认。这可能导致潜在严重的性能问题，特别是高速或可靠性较差的网络。

## 更多信息

还是以下图为例，服务器发送了4个片段并收到1条回复，确认号为201。因此，片段1和片段2被当成已确认。它们从重传队列中移出，同时允许服务器发送窗口向右移动200字节，从而发送数据增加200个字节。

然而，再次假设片段3，从sequence number201开始，在发送过程中丢失了。由于客户端从没有收到这一片段，所以它也无法发送确认号高于201的确认信息，从而导致滑动窗口停滞。服务器可以继续发送其他片段直到填满客户端的接收窗口，但是直到客户端发送另一条确认信息，服务器的发送窗口都不会滑动。

另一个问题是如果片段3丢失了，客户端将无法告知服务器是否收到后续的片段。在客户端接收窗口填满之前，很有可能客户端已经接收到片段4以及之后的片段。但是客户端无法发送值为501的确认信息以表明接收到片段4，因为这意味着片段3也接收到了。




这里我们看到了TCP单编号，累积确认机制的缺点。我们可以想象一个最差的情况，服务器被告知它有一个10，000字节窗口，20个片段每个片段500字节。第一个片段丢失了，其他19个被接收到了。但是由于第一个片段从没有接收到，其他19个也无法确认。

未确认片段处理策略：

我们怎样处理丢失片段之后的片段呢？本例中，当服务器片段3重传超时，它必须决定怎样处理片段4，它不知道客户端是否已经接收到。在上述最差情况下，第一个片段丢失后，其余19个可能或可能无法被客户端接收到。

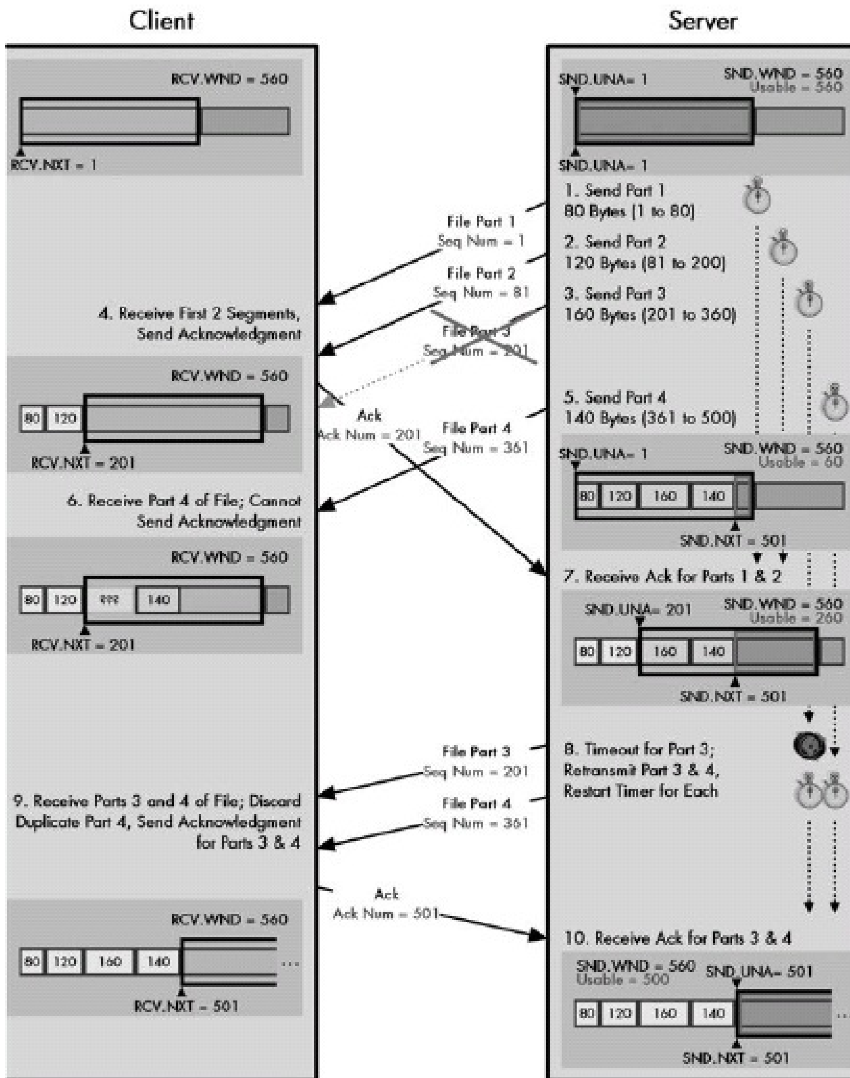
处理这种情况有两种可能的方式：

仅重传超时片段：这是一种更加保守的方式，仅重传超时的片段，希望其他片段都能够成功接收。如果该片段之后的其他片段实际上接收到了，这一方式是最佳的，如果没接收到，就无法正常执行。后者的情况每一个片段需要单独计时并重传。假设上述最坏情况下，所有20个500字节片段都丢失了。我们需要等片段1超时并重传。这一片段也许会得到确认，但之后我们需要等待片段2超时并重传。这一过程会重复多次。

重传所有片段：这是一种更激进或者说更悲观的方式。无论何时一个片段超时了，不仅重传该片段，还有所有其他尚未确认的片段。这一方式确保了任何时间都有一个等待确认的停顿时间，在所有未确认片段丢失的情况下，会刷新全部未确认片段，以使对端设备多一次接收

机会。在所有20个片段都丢失的情况下，相对于第一种方式节省了大量时间。这种方式的问题在于可能这些重传是不必要的。如果第一个片段丢失而其他19个实际上接收到了，也得重传那9500字节数据。

由于TCP不知道其他片段是否接收到，所以它也无法确认哪种方法更好，但只能选择一种方式。上图示例了保守的方式，而下图显示的是激进的方式：




问题的关键在于无法确认非连续片段。解决方式是对TCP滑动窗口算法进行扩展，添加允许设备分别确认非连续片段的功能。这一功能称为选择确认（selective acknowledgment, SACK）。

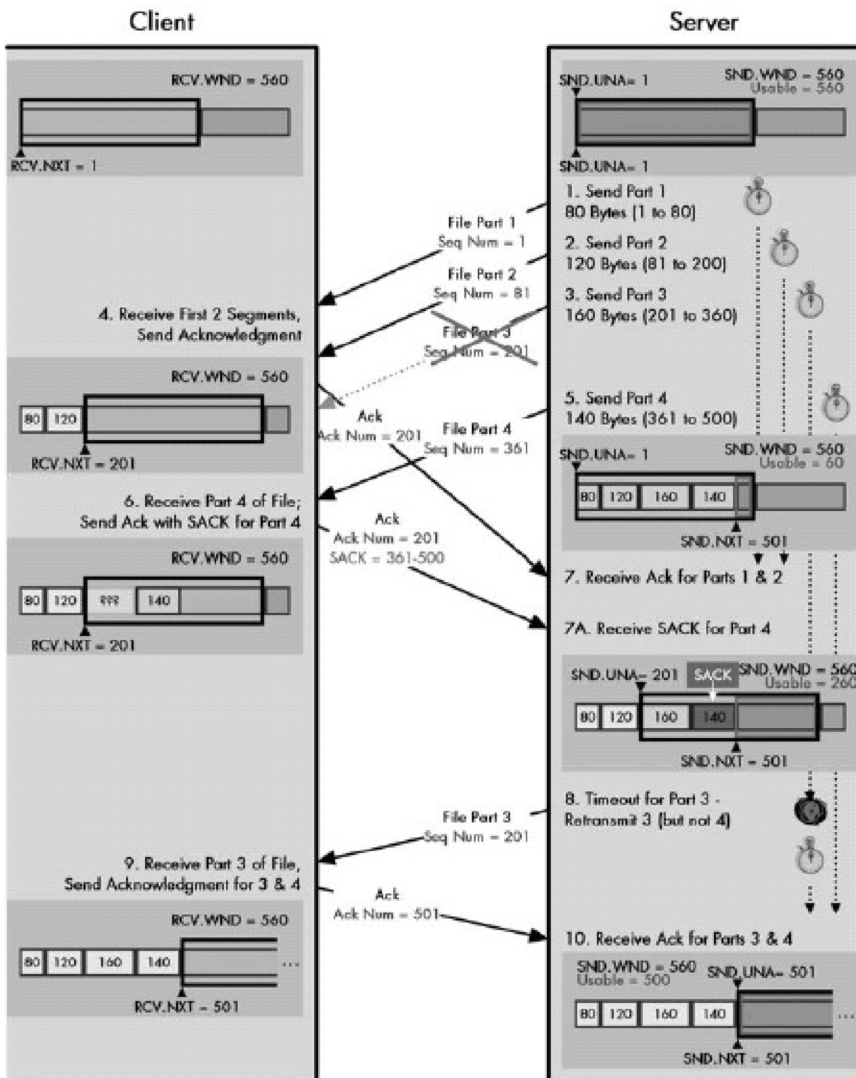
选择确认：

通过SACK，连接的两方设备必须同时支持这一功能，通过连接时使用的SYN片段来协商是否允许SACK。这一过程完成之后，任一设备都可以在常规TCP片段中使用SACK选项。这一选项包含一个关于已接收但未确认片段数据sequence number范围的列表，由于它们是非连续的。

各设备对重传队列进行修改，如果该片段已被选择确认过，则该片段中的SACK比特位置为1。该设备使用图2中激进方式的改进版本，一个片段重传之后，之后所有片段也会重传，除非**SACK**比特位为**1**。

例如，在4个片段的情况下，如果客户端接收到片段4而没有接收到片段3，当它发回确认号为201（片段1和片段2）的确认信息，其中包含一个SACK选项指明："已接收到字节361至500，但尚未确认"。如果片段4在片段1和2之后到达，上述信息也可以通过第二个确认片段来完成。服务器确认片段4的字节范围，并为片段4打开SACK位。当片段3重传时，服务器看到片段4的SACK位为1，就不会对其重传。如下图所示。

在片段3重传之后，片段4的SACK位被清除。这是为了防止客户端出于某种原因改变片段4已接收的想法。客户端应当发送确认号为501或更高的确认信息，正式确认片段3和4接收到。如果这一情况没有发生，服务器必须接收到片段4的另一条选择确认信息才能将它的SACK位打开，否则，在片段3重传时或计时器超时的情况下会对其自动重传。



# 网络基本功（十一）：TCP窗口调整与流控

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese



## 介绍

前文已经介绍过了TCP滑动窗口大小的重要性。在客户端与服务器的连接中，客户端告知服务器它一次希望从服务器接收多少字节数据，这是客户端的接收窗口，即服务器的发送窗口。类似地，服务器告知客户端一次希望从客户端接收多少字节数据，也就是服务器的接收窗口和客户端的发送窗口。

要理解为什么窗口大小会产生波动，首先需要理解它的含义。最简单的方式是它代表了设备对于特定连接的接收缓存大小。即，窗口大小代表一个设备一次能够从对端处理多少数据，之后再传递给应用层处理。

## 更多信息

当服务器从客户端接收数据，它就将数据放在缓存中，服务器必须对数据做以下两步操作：

确认：服务器必须将确认信息发回客户端以表明数据接收。

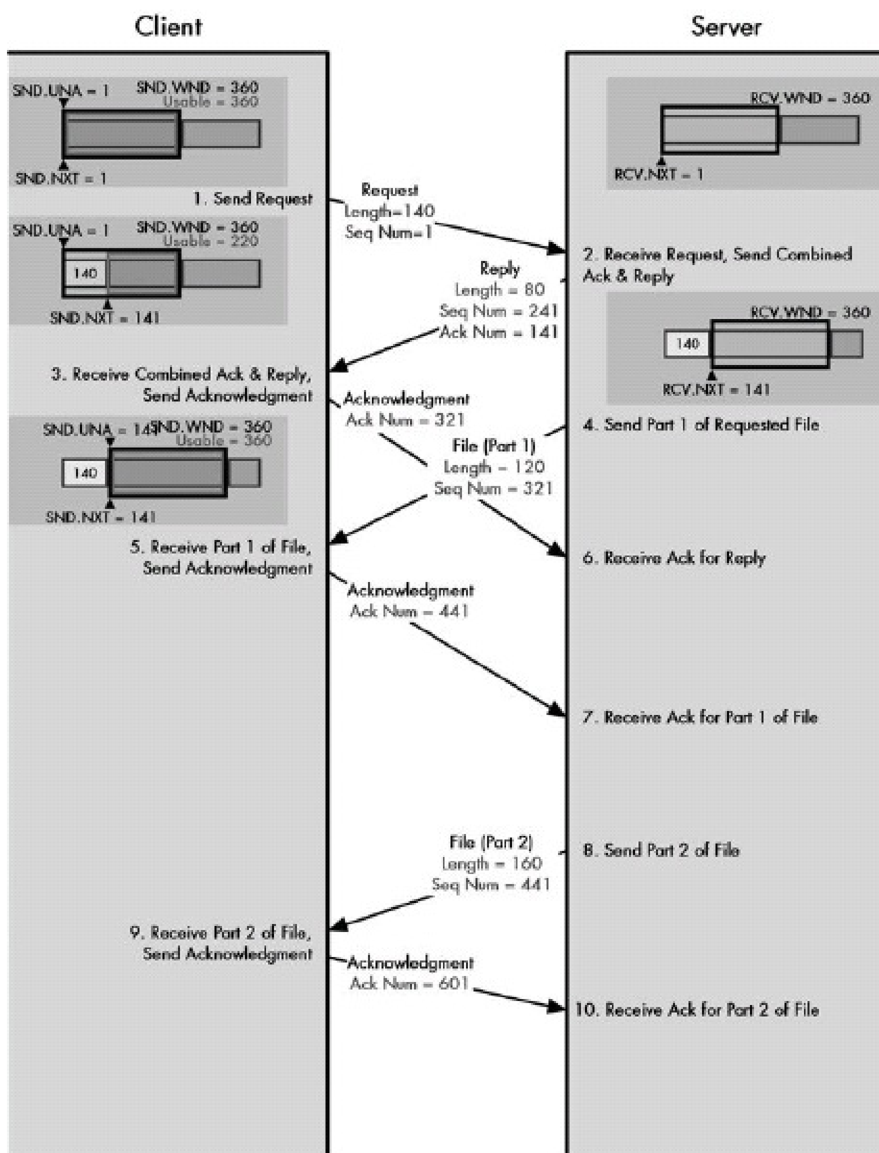
传输：服务器必须处理数据，将它传递给目标应用程序处理。

区分开这两件事情是非常重要的。关键在于基本的滑动窗口机制中，数据于接收时确认，但并不一定立即从缓存中传输出去。也就意味着当接收数据速度快于接收TCP处理速度时，缓存有可能被填满。当这一情况发生时，接收设备需要调整窗口大小已防止缓存过载。

由于窗口大小能够以这种方式管理连接两端设备数据流的速率，TCP就是以这种方式实现流控这一传输层非常典型的任务。流控对于TCP来说是很重要的，因为它是设备间互通状态的方式。通过增加或缩小窗口大小，服务器和客户端能够确保对端发送数据的速度等同于处理速度。

减小窗口大小以降低发送速率：

首先看一下客户端到服务器的数据传输，如下图所示。



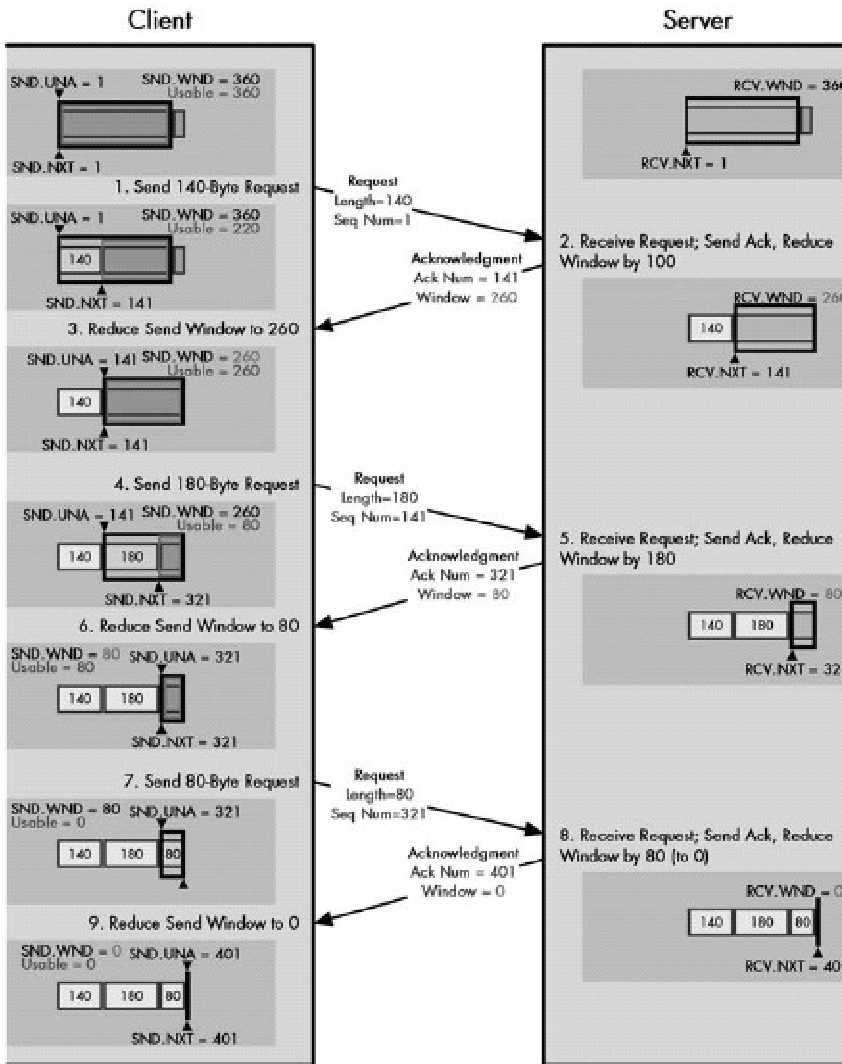

客户端传输140字节数据至服务器。之后，客户端的可用窗口还剩下220字节：发送窗口的360字节减去发送的140字节。

一段时间过后，服务器接收到140字节并将它们放在缓存中。现在，理想的情况下，140字节进入缓存，确认之后立刻从缓存移出。也就是说，缓存有足够的大小来容纳客户端发送的所有数据。缓存的空闲空间维持在360字节，因此告知客户端窗口大小保持不变。

只要服务器处理速度和数据进入速度相同，窗口大小就会保持在360字节。客户端在接收到140字节的确认信息以及窗口大小保持不变的信息之后，将360字节窗口向右移动140字节。由于现在未确认字节数为0，因此客户端又可以发送360字节数据。对应于之前可用窗口的220字节，加上刚刚确认的140字节数据。

然而，现实中服务器可能需要处理数十，数百乃至数千个TCP连接。TCP可能无法立刻处理数据，或应用应用程序本身无法接收140字节数据。任何一种情况下，服务器TCP都无法立刻将140字节从缓存中移出。这时，除了发回确认信息给客户端以外，服务器会想要告知客户端更改窗口大小，以表示缓存已经被部分写入了。

假设我们接收到140字节，但只能发送40字节给应用程序，缓存中剩下100字节。当发送140字节的确认信息，服务器将发送窗口缩小100字节，至260字节。当客户端从服务器接收到这一片段，它将会看到140字节的确认信息并将窗口向右滑动140字节。在滑动过程中，将大小缩减至260字节。可以认为将窗口左端滑动140字节，但右端仅滑动40字节。新的稍小一些的窗口保证服务器从客户端接收最多260字节数据，以适应接收缓存中的剩余空间，如下图的1-3步所示。




缩减发送窗口以停止发送新数据：

如果服务器无法接收任何新数据会怎么样呢？假设客户端下一次传输180字节，但是服务器太忙碌而无法对其进行处理。这种情况下，服务器将这180字节缓存下来，并且在确认信息中，将窗口大小从260字节缩减为80字节。当客户端接收到180字节的确认信息，它也会看到窗口缩减了180字节，它会滑动与缩减同样的大小，告知服务器：我确认接收180字节数据，但不允许你再发送新的数据。也可以看作窗口左端滑动180字节，但右端维持不动。只要右端不移动，客户端就无法发送更多数据。这一过程显示在上图的4-6中。

关闭发送窗口：

窗口调整可以通过双方设备来完成。如果服务器从客户端接收的数据持续快于推送给应用的速率，则服务器将会继续减小接收窗口。假设发送窗口减小至80字节，客户端发送第三个请求，长度为80字节，但服务器仍处于繁忙状态。之后服务器将窗口减小为0，也称为关闭窗口。这一信息告知客户端服务器已经过载，它需要彻底停止发送数据，如上图最后一步所示。之后，当服务器负载减轻时，可以再次增加这一连接的窗口，允许更多数据传输。

# 网络基本功（十二）：细说Linux网络配置（上）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

将一台设备添加到本地网络的基本步骤包括：

- 指定唯一的IP地址与主机名。
- 确保启动时正确配置网络接口。
- 创建默认路由。
- 指定DNS域名服务器以使设备能够连接到网络其他部分。

## 更多信息

指定主机名与**IP**地址**:**

使用**/etc/hosts**文件是将设备名映射到IP地址的最简单的方式，每一行以IP地址开始，跟随识别到的各种符号名：

```
127.0.0.1        localhost
192.108.21.48    lollipop.atrust.com lollipop loghost
192.108.21.254   chimchim-gw.atrust.com chimchim-gw
192.108.21.1     ns.atrust.com ns
192.225.33.5     licenses.atrust.com license-server
```

由于**/etc/hosts**仅包括本地映射而且必须维护在每一台客户端设备，所以最好保存那些需要在启动时映射的信息（即：主机本身，默认网关，以及域名服务器）。

可使用**hostname**命令为设备指定主机名。该命令通常在启动脚本中运行，脚本中包含从配置文件读取的主机名。

**ifconfig**：配置网络接口**:**

**ifconfig**打开或关闭网络接口，设置IP地址与子网掩码，以及其他选项和参数。通常在启动时通过命令行从配置文件中读取参数来运行，但也可以手动运行以做修改。

**ifconfig**命令格式如下：

**ifconfig** interface [family] address options…

例如：

**ifconfig eth0 192.168.1.13 netmask 255.255.255.0 up**

为eth0设置IPv4地址与子网掩码，并将该接口准备好供使用。大多数系统中， **ifconfig –a**列出系统的网络接口以及当前设置。

family参数告诉**ifconfig**配置的是哪一种网络协议。用户可以对一个接口设置多个协议并同时使用，但必须分开配置。IPv4选项为**inet**，IPv6选项为**inet6**。缺省为**inet**。

address参数指定接口的IP地址。也可以使用主机名，但该主机名必须能在启动时解析为IP地址。对于设备的主接口来说，这意味着主机名必须出现在本地**hosts**文件中，因为其他解析方式依赖于已被初始化的网络。

关键字**up**将接口打开，**down**将其关闭。

**netmask**选项为接口设置子网掩码。

**broadcast**选项为接口指定IP广播地址。

**route**：配置静态路由:

**route**命令指定静态路由，指明该路由表项永远不会更改，即使运行路由进程。当在本地网络中添加新的设备时，通常仅需要指定默认路由。

这里有两种情况：一，当报文目的地址是直连网络上的某台主机时，路由表中的 “next-hop gateway”是本地主机自己的一个接口，报文直接发送到目的地，这时可在配置接口时用**ifconfig**命令将路由添加到路由表中。 二，可能没有与目的地址相匹配的路由，这时，启用默认路由，否则，返回ICMP“network unreachable” 或 “host unreachable” 信息给发送方。很多本地网络只有一个出口，所以只需配置指向出口的默认路由。

每一条**route**命令添加或删除一条路由。如下**route**命令原型几乎适用于每一Linux版本：

**# route add -net 192.168.45.128/25 zulu-gw.atrust.net**

该命令通过网关路由器zulu-gw.atrust.net添加一条到192.168.45.128/25网络的路由。通常，网关路由器是相邻主机或本地主机的一个接口（Linux要求在网关地址前加上**gw**选项名）。**route**命令必须能够将zulu-gw.atrust.net解析成IP地址。

**Ubuntu**网络配置:

如下图所示，Ubuntu在**/etc/hostname**以及**/etc/network/interfaces**，以及**/etc/network/options**中配置网络信息。

| File | What's set there |
|---|---|
| hostname | Hostname |
| network/interfaces | IP address, netmask, default route |

主机名在 **/etc/hostname**中设置。很多场景都要用到这一文件中配置的名字，某些情况下对命名是有限制要求的。

IP地址，网络掩码，默认网关在 **/etc/network/interfaces**中设置。以iface关键字开头的一行介绍了各个接口。iface之后的缩进行指明附加参数。例如：

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
    address 192.168.1.102
    netmask 255.255.255.0
    gateway 192.168.1.254
```

**ifup**和**ifdown**命令会读取该文件并通过调用下层命令（诸如**ifconfig**）并配以合适的参数将接口连通或断开。auto语句指定启动时默认或ifup –a运行时的连通接口。

iface行中的inet关键字是ifconfig中使用的地址。关键字static表示一种“方式”，指eth0的IP地址和网络掩码是直接指定的。地址和网络掩码行要求静态配置，gateway行指明默认网关，用于安装默认路由。

**SUSE**网络配置：

SUSE用户可以选择NetworkManager或是传统的配置方法，用户可以在YaST中做出选择。也可以使用YaST GUI来配置传统系统。这里，我们介绍传统方式。除了配置网络接口以外，YaST也提供 **/etc/hosts**文件，静态路由，DNS配置的直接UI。下图显示了底层的配置文件。

| File | What's set there |
| --- | --- |
| **ifcfg**-*interface* | Hostname, IP address, netmask, and more |
| **ifroute**-*interface* | Interface-specific route definitions |
| **routes** | Default route and static routes for all interfaces |
| **config** | Lots of less commonly used network variables |

除了DNS参数以及系统主机名之外，SUSE将大多数网络选项配置在 **/etc/sysconfig/network** 目录下的**ifcfg**-interface文件。每一个接口呈现一个文件。

除了指定接口的IP地址，网关，以及广播信息，**ifcfg-***文件可以配置很多其他网络选项。**ifcfg.template**文件对很多参数有清楚的注释。以下图为例：

```
BOOTPROTO='static'       # Static is implied but it doesn't hurt to be verbose.
IPADDR='192.168.1.4/24'  # The /24 defines the NETWORK and NETMASK vars
NAME='AMD PCnet - Fast 79C971' # Used to start and stop the interface.
STARTMODE='auto'         # Start automatically at boot
USERCONTROL='no'         # Disable control through kinternet/cinternet GUI
```

SUSE系统中全局静态路由信息（包括默认路由）存储在**routes**文件中。文件中的各行就好象**route**命令省略了选项名，内容包含目标地址，网关，掩码，接口以及可选参数存储在路由表中，供路由进程查询。对于上述仅有默认路由的主机来说，路由文件包含以下内容：

default 192.168.1.254 - -

针对不同接口的路由保存在**ifroute**-interface文件中，接口部件的命名方法与**ifcfg-***文件一致。其内容格式与**routes**文件相同。

**Red Hat**网络配置：

Red Hat网络配置GUI名为**system-config-network**，也可以通过Network名下面的System->Administration菜单下访问。该工具为配置网络接口与静态路由提供了一个简单的UI，也提供建立IPsec通道，配置DNS，添加**/etc/hosts**的面板。

下表列出了GUI编辑的底层文件。可在**/etc/sysconfig/network**中设置机器的主机名，也包括DNS域名以及默认网关。

| File | What's set there |
| --- | --- |
| **network** | Hostname, default route |
| **static-routes** | Static routes |
| **network-scripts/ifcfg-*ifname*** | Per-interface parameters: IP address, netmask, etc. |

例如，以下是某个以太网接口的**network**文件：

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=redhat.toadranch.com
DOMAINNAME=toadranch.com        ### optional
GATEWAY=192.168.1.254
```

接口相关的数据存储在 **/etc/sysconfig/network-scripts/ifcfg-**ifname，ifname是网络接口的名字，该文件为每个接口设置IP地址，掩码，网络，以及广播地址。也包含指明接口是否要在启动时开启。

常规机器有配置以太网接口以及回环接口的文件，例如

```
DEVICE=eth0
IPADDR=192.168.1.13
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
```

以及

```
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

基于DHCP的echo文件更加简单：

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

在**/etc/sysconfig**文件中更改配置信息之后，对相应端口运行 **ifdown** ifname， **ifup** ifname。如果一次配置多个端口，使用命令**service network restart**重置整个网络。这其实是运行**/etc/rc.d/init.d/network**的一个快速的方法，每次启动时被调用，加上start参数。

也可以通过启动脚本来配置静态路由，添加到**/etc/sysconfig/static-routes** 文件的路由信息在启动时被存入路由表。这些表项为**route add**命令指定参数：

```
eth0 net 130.225.204.48 netmask 255.255.255.248 gw 130.225.204.49
eth1 net 192.38.8.0 netmask 255.255.255.224 gw 192.38.8.129
```

首先列出的是接口，但它实际上是在**route**命令行的最后执行，将路由与指定接口相关联。（也可以在GUI中看到该架构，路由作为部分设置内容配给各个接口）。命令剩下的内容包含**route**参数。上文静态路由的例子会产生如下命令：

```
route add -net 130.225.204.48 netmask 255.255.255.248 gw 130.225.204.49 eth0
route add -net 192.38.8.0 netmask 255.255.255.224 gw 192.38.8.129 eth1
```

# 网络基本功（十三）：细说Linux网络配置（下）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

本文承接细说Linux网络配置（上）。

## 更多信息

**Linux**网络硬件选项:

**ethtool**命令查询并设置网络接口关于媒体相关的参数。如：链路速度和双工。它代替了以前的**mii-tool**命令，但有些系统中两者并存。

只要简单加上接口名就可以查询它的状态。例如，eth0接口（PC主板的网卡接口）启动了自协商并且运行于全速率：

```
ubuntu# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half    10baseT/Full
                                100baseT/Half   100baseT/Full
                                1000baseT/Half  1000baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half    10baseT/Full
                                100baseT/Half   100baseT/Full
                                1000baseT/Half  1000baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000033 (51)
        Link detected: yes
```

将该接口锁定在100 Mb/s全双工，使用以下命令：

```
ethtool -s eth0 speed 100 duplex full
```

如果想知道自协商在系统中是否可靠，也可以使用**ethtool –r**命令，可使链路参数立刻重新协商。

另一个有用的选项是**-k**，显示哪些协议相关任务指定给网络接口，而不是由内核执行。大多数接口能够计算校验和（checksum），一些也可以辅助分段任务。可以通过**ethtool –K**命令结合多个子选项开启或禁用特定类型的offloading（-k显示当前值，-K对其进行设置）。

通过**ethtool**所做的变更是暂时的。如果希望永久性更改，需要确保**ethtool**作为系统网络配置的一部分来运行。最好是把它作为各个接口配置的一部分，如果你只是在启动时运行一些**ethtool**命令，那么在接口重启而系统未重启时配置就无法正确生效。

注：Red Hat系统中，可以在**/etc/sysconfig/network-scripts. ifup**下的配置文件中添加一行ETHTOOL_OPTS=，以将整行作为参数传递给**ethtool**。

SUSE中ethtool的用法与Red Hat相似，但是选项名为ETHTOOL_OPTIONS，配置文件保存在**/etc/sysconfig/network**。

Ubuntu系统中，可以在 **/etc/network/interfaces**的接口配置脚本中运行**ethtool**命令。

**Linux TCP/IP**选项：

Linux将每个可调内核变量放在**/proc**虚拟文件系统中。网络变量位于**/proc/sys/net/ipv4**。以下是一些重要变量的列表：

```
ubuntu$ cd /proc/sys/net/ipv4; ls -F
...                             ...                       tcp_no_metrics_save
conf/                           tcp_congestion_control    tcp_orphan_retries
icmp_echo_ignore_all            tcp_dma_copybreak         tcp_reordering
icmp_echo_ignore_broadcasts     tcp_dsack                 tcp_retrans_collapse
...                             tcp_ecn                   tcp_retries1
icmp_ratelimit                  tcp_fack                  tcp_retries2
icmp_ratemask                   tcp_fin_timeout           tcp_rfc1337
igmp_max_memberships            tcp_frto                  tcp_rmem
igmp_max_msf                    tcp_frto_response         tcp_sack
inet_peer_gc_maxtime            tcp_keepalive_intvl       ...
inet_peer_gc_mintime            tcp_keepalive_probes      tcp_stdurg
inet_peer_maxttl                tcp_keepalive_time        tcp_synack_retries
inet_peer_minttl                tcp_low_latency           tcp_syncookies
inet_peer_threshold             tcp_max_orphans           tcp_syn_retries
ip_default_ttl                  tcp_max_ssthresh          tcp_timestamps
ip_dynaddr                      tcp_max_syn_backlog       ...
ip_forward                      tcp_max_tw_buckets        udp_mem
...                             tcp_mem                   udp_rmem_min
neigh/                          tcp_moderate_rcvbuf       udp_wmem_min
route/                          tcp_mtu_probing
```

许多名字中含有rate和max的变量用作阻止服务器攻击。子目录**conf**包含按照各接口设置的变量，包括**all**和**default**以及各接口子目录（包括loopback）。各子目录包含相同的一组文件。

```
ubuntu$ cd conf/default; ls -F
accept_redirects        disable_policy          promote_secondaries
accept_source_route     disable_xfrm            proxy_arp
arp_accept              force_igmp_version      rp_filter
arp_announce            forwarding              secure_redirects
arp_filter              log_martians            send_redirects
arp_ignore              mc_forwarding           shared_media
bootp_relay             medium_id               tag
```

假设用户在**conf/eth0**子目录中更改了一个变量，则变更仅适用于该接口。如果在**conf/all**中更改了变量值，你也许认为更改适用于所有接口，但实际上并非如此。每一个变量对于接收通过**all**所作的更改有各自的规则。有些是与当前值做或运算，有些是做与运算，还有些是取最大或最小值。除了内核代码以外没有文档详细说明这一过程，因此最好避免这样做，比较好的做法是对各接口分别做修改。

如果用户在**conf/default**中修改了变量，新的值会传递到所有在这之后配置的接口。另一方面，最好保持默认值不变，以供取消更改时参考。

**/proc/sys/net/ipv4/neigh**目录同样包含了各接口子目录。子目录中的文件掌控相应接口的ARP table管理以及IPv6邻居发现。以下是变量列表，以gc（代表垃圾回收）开头的变量决定ARP table表项超时以及丢弃。

```
ubuntu$ cd neigh/default; ls -F
anycast_delay             gc_stale_time     proxy_delay
app_solicit               gc_thresh1        proxy_qlen
base_reachable_time       gc_thresh2        retrans_time
base_reachable_time_ms    gc_thresh3        retrans_time_ms
delay_first_probe_time    locktime          ucast_solicit
gc_interval               mcast_solicit     unres_qlen
```

要查看变量值，使用**cat**命令，要进行设置，使用**echo**重定向到合适的文件名。例如：

```
ubuntu$ cat icmp_echo_ignore_broadcasts0
```

显示当变量值为0时，则广播ping不能被忽略。要将它设置为1，在 **/proc/sys/net** 中，运行

```
ubuntu$ sudo sh -c "echo 1 &gt; icmp_echo_ignore_broadcasts"
```

通常，你登录的网络与调整的网络是同一个，所以要小心行事。在更改生产设备配置前务必在台式机上测试。

要永久更改某参数（更准确的说，系统每次启动时都重置该值），在**/etc/sysctl.conf**中添加合适的变量，这些变量在启动时由**sysctl**命令读取。文件**sysctl.conf**的格式是变量名__=值，而不是手动在shell中修改的格式**echo value > variable**。变量名是相对于**/proc/sys**的路径，可以用点或斜杠。例如：

**/etc/sysctl.conf** 文件中，

```
net.ipv4.ip_forward=0
net/ipv4/ip_forward=0
```

都会将主机IP转发关闭。

同时，内核源版本中的 **ip-sysctl.txt**文件也有一些比较好的注释信息。

# 参考

Unix and Linux System Administration Handbook

# 网络基本功（十四）：细说诊断工具ping

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

ping的工作原理很简单，一台网络设备发送请求等待另一网络设备的回复，并记录下发送时间。接收到回复之后，就可以计算报文传输时间了。只要接收到回复就表示连接是正常的。耗费的时间喻示了路径长度。重复请求响应的一致性也表明了连接质量的可靠性。因此，ping回答了两个基本的问题：是否有连接？连接的质量如何？本文主要讨论这两个问题。

## 更多信息

正常的ping操作主要是两个特定的ICMP消息，ECHO_REQUEST和ECHO_REPLY。理论上，所有TCP/IP网络设备都应当通过返回报文来响应ECHO_REQUEST，但实际上并不总是如此。

**ping**的解析**:**

大多数操作系统版本，会一直发送ECHO_REQUESTs，直到中断为止。例如：

```
bsd1# ping www.bay.com
PING www.bay.com (204.80.244.66): 56 data bytes
64 bytes from 204.80.244.66: icmp_seq=0 ttl=112 time=180.974 ms
64 bytes from 204.80.244.66: icmp_seq=1 ttl=112 time=189.810 ms
64 bytes from 204.80.244.66: icmp_seq=2 ttl=112 time=167.653 ms
^C
--- www.bay.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 167.653/179.479/189.810/9.107 ms
bsd1#
```

这一过程被Ctrl-C中断，此时打印出汇总统计。

上述结果中，针对每一个报文的回复给出了报文大小，来源，ICMP sequence number，TTL值，以及往返时间。其中，sequence number和往返时间对于评估基本连接状况来说是最有用的信息。

当发送一个ECHO_REQUEST时，将发送时间记录在报文里，并复制到远端主机相应的ECHO_REPLY报文中。当接收到ECHO_REPLY时，通过比较当前时间与报文时间计算出耗费时间。如果没有收到符合该sequence number的报文，则认为该报文丢失。耗费时间长短以及变化范围取决于中间链路数量，速度，以及链路拥塞情况。

什么值是合理的呢？这一值高度取决于网络以及网络质量。如果是在LAN网络环境下，响应时间还是很快的，通常在十几毫秒范围之内。如果连接到外网则该值会显著增加。如果是远程站点，可能会耗时上百毫秒。

你也可以用ping来粗略计算连接的吞吐量。在外网上发送两个不同大小的报文，通过-s选项来完成。时间长度的差别会反映大报文中额外数据所耗费的时间。例如，假设ping 100字节耗费30ms而ping 1100字节耗费60ms，因此，往返额外花费30ms单程额外花费15ms，多发送1000字节或8000比特。吞吐量近似为每15ms 8000比特或540，000bps。两个测量值之间的差异用来扣除开销。当然这一估算是非常粗略的，没有考虑到路径上其他数据流的情况，也没有考虑路径上所有链路的情况。

TTL貌似可以估算一条路径上的跳数，但是这有一些问题。当发送报文时，TTL字段先被初始化接着经过路径上每个路由器都要递减。如果达到0，报文就被丢弃了。从而对所有报文生命周期有一定限制。因而在路由回环的过程中，报文不会无期限存在于网络上。不幸的是，TTL字段可能会，也可能不会被远端设备重置，如果重置，也没有一致性。因此，要使用TTL字段估算路径中的跳数需要知道详细的系统信息。

通常一串稳定的回复意味着健康的连接。如果报文丢失或丢弃，可以在sequence number中看到跳数，以及丢失报文的编号。偶尔丢失一个报文不表示真的有什么问题。特别是跨越多台路由器或拥塞网络时。一个序列中的一个报文丢失或耗费明显更长时间是很正常的，这是因为路径中各条链路需对第一个报文做ARP解析。在ARP数据保存之后，后续报文就不会有这种开销。但是，如果丢失报文比例较大，则有可能路径上有问题。

某些情况下会收到ICMP错误消息。通常来自路由器，这里面包含很有用的信息。例如，下例中，设备尝试访问一个不存在的网络上的设备：

```
bsd1# ping 172.16.4.1
PING 172.16.4.1 (172.16.4.1): 56 data bytes
36 bytes from 172.16.2.1: Destination Host Unreachable
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
4  5  00 5400 5031   0 0000  fe  01 0e49 172.16.2.13  172.16.4.1
36 bytes from 172.16.2.1: Destination Host Unreachable
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
4  5  00 5400 5034   0 0000  fe  01 0e46 172.16.2.13  172.16.4.1
^C
--- 172.16.4.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

由于路由器没有到达该网络的路径，所以返回ICMP DESTINATION_HOST_UNREACHABLE信息。通常如果问题发生在运行ping命令的设备上，则会收到Destination Host Unreachable告警或 Destination Network Unreachable告警。如果问题发生在转发报文的设备上，则只会收到一条Destination Host Unreachable。

下例中，尝试向一台已配置拒绝从源设备接收数据流的路由器发送数据：

```
bsd1# ping 172.16.3.10
PING 172.16.3.10 (172.16.3.10): 56 data bytes
36 bytes from 172.16.2.1: Communication prohibited by filter
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
4  5  00 5400 5618   0 0000  ff  01 0859 172.16.2.13  172.16.3.10
36 bytes from 172.16.2.1: Communication prohibited by filter
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
4  5  00 5400 561b   0 0000  ff  01 0856 172.16.2.13  172.16.3.10
^C
--- 172.16.3.10 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
被过滤条件阻止的告警信息表明报文被丢弃。但也有可能过滤条件不显示该告警。
下例中，
bsd1# ping 172.16.3.10
PING 172.16.3.10 (172.16.3.10): 56 data bytes
^C
--- 172.16.3.10 ping statistics ---
6 packets transmitted, 0 packets received, 100% packet loss
```

路由器上使用同样的过滤条件，但应用于离开网络的数据流，而不作用于inbound数据流。因此，没有消息发送。这时，ping就无法告诉你为什么报文没有收到回复。

**ping**的选项**:**

一些选项控制发送报文的速率和数量，-c选项允许用户指定发送报文的数量。例如，ping –c10会发送10个报文然后停止。这一命令在脚本中很有用处。

命令-f和-l用于将报文泛洪到网络上。-f选项表明报文发送速率与接收主机能够处理速率相同。这一参数可用于链路压力测试或接口性能比较。

-l选项用于计数，尽可能快的发送该数量报文，然后恢复正常。该命令用于测试处理泛洪的能力，需要root权限执行。

-i选项用于用户在两个连续报文之间指定等待秒数。该命令对于将报文间隔开或用在脚本中非常有用。正常情况下，偶然的ping包对数据流的影响是很小的。但重复报文或报文泛洪影响就很大了。因此，使用以上选项时需谨慎。

-n选项将输出限制为数字形式，这在碰见DNS问题时很有用。-v显示更详尽输出，较少输出为-q和-Q。

-s选项指定发送数据的大小。但如果设置的太小，小于8，则报文中就没有空间留给时间戳了。设置报文大小能诊断有路径MTU(Maximum Transmission Unit)设置或分段而导致的问题。如果不使用该选项，ping默认是64字节。

# 参考

Network Troubleshooting Tools

# 网络基本功（十五）：细说网络性能监测与实例（上）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

网络路径性能检测主要包括三方面的内容：带宽测量能够获知网络的硬件特性，如网络的最大容量，吞吐量测量能够获得网络实际可提供的最大容量，数据流测量能够了解真实占用的网络容量。

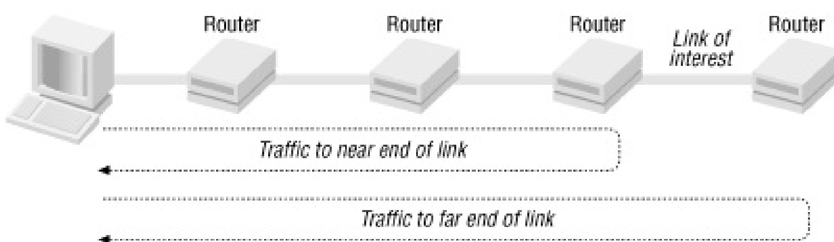
本文介绍在评估网络性能是否合理时，需要收集的数据及收集方式。涉及工具包括：**ping, pathchar, bing, ttcp, netperf, iperf, netstat**。

## 更多信息

带宽测量:

**ping**

ping这一工具返回的时间，虽然通常被描述为传输延时，实际上是发送，传输，队列延时之和。上一节中，我们通过ping来粗略计算带宽。这一过程可通过如下方式改进：首先计算链路近端的路径行为，然后计算远端路径，然后用两者差异来估算链路带宽。




这一过程需要四次使用ping。首先，用两个不同大小报文ping近端链路。减掉传输大报文中额外数据的传输时间以外，时间差可估算传输以及队列延时。接下来，用同样两个报文ping远端链路。再次用大报文和小报文的时间差来估算开销。最后，用两次差值的差值就是在最后一段链路中传输额外数据的时间值。这是一个往返时间，除以2就是额外数据在单向链路传输所用时间。带宽则是额外数据总量除以单向传输时间。

下表是第二跳和第三跳的时间值，报文大小为100和1100字节。

| IP address | Time for 100 bytes | Time for 1100 bytes |
|---|---|---|
| 205.153.61.1 | 1.380 ms | 5.805 ms |
| 205.153.60.2 | 4.985 ms | 12.823 ms |
| 165.166.36.17 | 8.621 ms | 26.713 ms |

下表显示了带宽计算结果，用time difference除以2，用8000bit除以这个值，再乘1000（毫秒转换为秒）。结果是bps转换为Mbps。

| Near link | Far link | Time difference | Estimated bandwidth |
|---|---|---|---|
| 205.153.61.1 | 205.153.60.2 | 3.413 ms | 4.69 Mbps |
| 205.153.60.2 | 165.166.36.17 | 10.254 ms | 1.56 Mbps |

### pathchar

将上述过程自动话完成的一个工具是pathchar。pathchar在路径的一端即能检测各链路的带宽。方法与之前描述的ping相类似，但是pathchar使用各种大小不一的报文。如下例所示：

```
bsd1# pathchar 165.166.0.2
pathchar to 165.166.0.2 (165.166.0.2)
mtu limited to 1500 bytes at local host
doing 32 probes at each of 45 sizes (64 to 1500 by 32)
0 205.153.60.247 (205.153.60.247)
|   4.3 Mb/s,   1.55 ms (5.88 ms)
1 cisco (205.153.60.2)
|   1.5 Mb/s,   -144 us (13.5 ms)
2 165.166.36.17 (165.166.36.17)
|    10 Mb/s,   242 us (15.2 ms)
3 e0.r01.ia-gnwd.Infoave.Net (165.166.36.33)
|   1.2 Mb/s,   3.86 ms (32.7 ms)
4 165.166.125.165 (165.166.125.165)
|   ?? b/s,   2.56 ms (37.7 ms)
5 165.166.125.106 (165.166.125.106)
|    45 Mb/s,   1.85 ms (41.6 ms),  +q 3.20 ms (18.1 KB) *4
6 atm1-0-5.r01.ncchrl.infoave.net (165.166.126.1)
|    17 Mb/s,   0.94 ms (44.3 ms),  +q 5.83 ms (12.1 KB) *2
7 h10-1-0.r01.ia-chrl.infoave.net (165.166.125.33)
|   ?? b/s,   89 us (44.3 ms),  1% dropped
8 dns1.InfoAve.Net (165.166.0.2)
8 hops, rtt 21.9 ms (44.3 ms), bottleneck 1.2 Mb/s, pipe 10372 bytes
```

pathchar的运行过程中，首先显示的信息描述探测如何进行。从第三行输出开始，可看到pathchar使用从64到1500字节的45中不同大小报文。对于每一跳使用32种不同报文组合进行测试。因此，共8跳生成了11，520个测试报文加上相应回复信息。

显示中给出了带宽和延时。pathchar也包括了队列延时信息（如本例中5和6）。如上述信息，pathchar并不总是能成功估算出带宽（如链路4和7）或是延时（如链路1）。

在pathchar运行过程中，每发送一个报文就启动一次倒计时：显示内容如下所示：

```
1:  31   288   0       3
```

1指示跳数并且随着路径上后续跳数而增加。下一个数字是倒计时值，给出这一链路剩余的探测组数。第三个值是当前发送报文大小。第二个和第三个值改变都非常迅速。倒数第二个值是目前为止丢弃报文数，最后一个是该链路的平均往返时间。

当一条的探测完成时，这一行内容被带宽，传输延时，往返时间所取代。pathchar使用观测到的最小延时来改进带宽估算值。

**bing**

pathchar的一个替代工具是bing。pathchar估算的是一条路径上各链路的带宽，而bing用来测量点到点的带宽。通常，如果你不知道路径上的各条链路，需要首先执行traceroute命令。之后可以运行bing来指定链路的近端和远端。下例显示了第三跳的带宽：

```
bsd1# bing -e10 -c1 205.153.60.2 165.166.36.17
BING    205.153.60.2 (205.153.60.2) and 165.166.36.17 (165.166.36.17)
        44 and 108 data bytes
1024 bits in 0.835ms: 1226347bps, 0.000815ms per bit
1024 bits in 0.671ms: 1526080bps, 0.000655ms per bit
1024 bits in 0.664ms: 1542169bps, 0.000648ms per bit
1024 bits in 0.658ms: 1556231bps, 0.000643ms per bit
1024 bits in 0.627ms: 1633174bps, 0.000612ms per bit
1024 bits in 0.682ms: 1501466bps, 0.000666ms per bit
1024 bits in 0.685ms: 1494891bps, 0.000669ms per bit
1024 bits in 0.605ms: 1692562bps, 0.000591ms per bit
1024 bits in 0.618ms: 1656958bps, 0.000604ms per bit
--- 205.153.60.2 statistics ---
bytes   out    in   dup  loss   rtt (ms): min       avg       max
   44    10    10          0%           3.385     3.421     3.551
  108    10    10          0%           3.638     3.684     3.762
--- 165.166.36.17 statistics ---
bytes   out    in   dup  loss   rtt (ms): min       avg       max
   44    10    10          0%           3.926     3.986     4.050
  108    10    10          0%           4.797     4.918     4.986
--- estimated link characteristics ---
estimated throughput 1656958bps
minimum delay per packet 0.116ms (192 bits)
average statistics (experimental) :
packet loss: small 0%, big 0%, total 0%
average throughput 1528358bps
average delay per packet 0.140ms (232 bits)
weighted average throughput 1528358bps
resetting after 10 samples.
```

输出从地址和报文大小信息开始，之后是探测pair。接下来，返回往返时间和丢失数据。最后，返回一些吞吐量的估测值。

吞吐量测量:

吞吐量不够的原因不仅在于硬件不足，还有可能是网络设计架构的问题。例如，广播域设置得太大，则即使硬件够磅也会造成问题。解决方案是重构网络，在充分理解数据流模式后，将这类域隔离开或是分段。

吞吐量通常是测量大块数据传输延时来完成的。通常需要在链路各端运行软件。一般这类软件运行在应用层，所以它不仅测量网络也测量了软硬件。

一个比较简单粗放的方式是用FTP。用FTP来传输一份文件并且看一下它report的数据。需要将结果转换成比特率，例如，这是文件传输的最后一行：

```
1294522 bytes received in 1.44 secs (8.8e+02 Kbytes/sec)
```

将1,294,522字节乘8转换成bit之后再除以时间，1.44秒。 结果为7,191,789 bps。

这种方法的不足在于磁盘访问时间可能对结果造成影响。如果需要提高精度则需要使用一些工具。

### ttcp

运行这一程序首先需要在远端设备运行server，通常用-r和-s选项。之后运行client，用-t和-s选项，以及主机名或地址。数据从client端发送至server端，测量性能之后，在各端返回结果，之后终止client端和server端。例如，server端如下所示：

```
bsd2# ttcp -r -s
ttcp-r: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp
ttcp-r: socket
ttcp-r: accept from 205.153.60.247
ttcp-r: 16777216 bytes in 18.35 real seconds = 892.71 KB/sec +++
ttcp-r: 11483 I/O calls, msec/call = 1.64, calls/sec = 625.67
ttcp-r: 0.0user 0.9sys 0:18real 5% 15i+291d 176maxrss 0+2pf 11478+28csw
client端如下所示：
bsd1# ttcp -t -s 205.153.63.239
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp  -> 205.153.63.239
ttcp-t: socket
ttcp-t: connect
ttcp-t: 16777216 bytes in 18.34 real seconds = 893.26 KB/sec +++
ttcp-t: 2048 I/O calls, msec/call = 9.17, calls/sec = 111.66
ttcp-t: 0.0user 0.5sys 0:18real 2% 16i+305d 176maxrss 0+2pf 3397+7csw
```

该程序报告中显示了信息传输总量，标识了连接的建立，并且给出了结果，包括raw data，throughput，I/O call信息，执行时间。最有用的信息应该是transfer rate，892.71 KB/sec (or 893.26 KB/sec)。

这一数据反映了数据的传输速率，而不是链路的容量。将这一数据转化成带宽可能是有问题的，因为实际上传输了比这一值更多的比特数。这一程序显示18.35秒传送了16,777,216字节，但是这仅仅是数据。以太网报文封装还包括TCP，IP，以太网报文头，估算容量时，需要把这些值加上去。

吞吐量低通常意味着拥塞，但也并不总是如此。吞吐量也会取决于配置问题，如连接的TCP窗口大小。如果窗口大小不足，会严重影响到性能。

（未完待续）

# 参考

Network Troubleshooting Tools

# 网络基本功（十六）：细说网络性能监测与实例（下）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

网络问题中，性能问题是最复杂的问题之一，解决这样的问题能够透彻的了解整个网络的结构。但通过合适的吞吐量和数据流测试工具，能够帮你快速找到问题所在。本文承接上文，阐述netperf和netstat的用法。

## 更多信息

吞吐量测量:

(承接上文)

**netperf**

该程序是由HP创造，该程序免费可用，运行于一些Unix平台，有支持文档，也被移植到Windows平台。虽然不像ttcp那样无处不在，但它的测试范围更加广泛。

与ttcp不同，客户端和服务器端是分开的程序。服务器端是netserver，能够单独启动，或通过inetd启动。客户端是netperf。下例中，服务器和客户端启动于同一台机器：

```
bsd1# netserver
Starting netserver at port 12865
bsd1# netperf
TCP STREAM TEST to localhost : histogram
Recv   Send     Send
Socket Socket  Message  Elapsed
Size   Size     Size     Time      Throughput
bytes  bytes    bytes    secs.     10^6bits/sec
16384  16384  16384    10.00     326.10
```

测试的是loop-back接口，报告显示吞吐量为326Mbps。

下例中，netserver启动于主机：

```
bsd1# netserver
Starting netserver at port 12865
netperf加上-H选项指定服务器地址：
bsd2# netperf -H 205.153.60.247
TCP STREAM TEST to 205.153.60.247 : histogram
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
16384  16384  16384    10.01       6.86
```

大致与ttcp所得出的吞吐量相同。netperf还进行了一些额外的测试。以下测试中，还计算了连接的transaction rate：

```
bsd2# netperf -H 205.153.60.247 -tTCP_RR
TCP REQUEST/RESPONSE TEST to 205.153.60.247 : histogram
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
16384  16384  1        1       10.00     655.84
16384  16384
```

该程序包含一些测试脚本。也可以使用netperf做各种流测试。

**iperf**

如果ttcp和netperf都不符合你的要求，那么可以考虑iperf。iperf也可以用于测试UDP带宽，丢失率，和抖动。Java前端让该工具便于使用。该工具同样移植入windows。

下例是运行iperf服务器端：

```
bsd2# iperf -s -p3000
------------------------------------------------------------
Server listening on TCP port 3000
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  4] local 172.16.2.236 port 3000 connected with 205.153.63.30 port 1133
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   5.6 MBytes   4.5 Mbits/sec
^C
```

下例是在windows运行客户端：

```
C:\>iperf -c205.153.60.236
-p3000
------------------------------------------------------------
Client connecting to 205.153.60.236, TCP port 3000
TCP window size:  8.0 KByte (default)
------------------------------------------------------------
[ 28] local 205.153.63.30 port 1133 connected with 205.153.60.236 port 3000
[ ID] Interval       Transfer     Bandwidth
[ 28]  0.0-10.0 sec   5.6 MBytes   4.5 Mbits/sec
```

注意使用Ctrl-C来终止服务器端。在TCP模式下，iperf相当于ttcp，所以它可盈用户客户端或服务器。

在研究TCP窗口是否足够大时，使用iperf特别方便。-w选项设置socket buffer大小。对于TCP来说，这就是窗口大小。通过-w选项，用户可以单步调试各种窗口大小来看它们是怎样影响吞吐量的。

其他工具

你也许想要考虑一些相关或类似的工具。treno使用的方法类似于traceroute来计算块容量，路径MTU，以及最小RTP。如下例所示：

```
bsd2# treno 205.153.63.30
MTU=8166  MTU=4352  MTU=2002  MTU=1492 ..........
Replies were from sloan.lander.edu [205.153.63.30]
    Average rate: 3868.14 kbp/s (3380 pkts in + 42 lost = 1.2%) in 10.07 s
Equilibrium rate:      0 kbp/s (0 pkts in + 0 lost =   0%) in    0 s
Path properties: min RTT was  13.58 ms, path MTU was 1440 bytes
XXX Calibration checks are still under construction, use –v
```

通常来说，netperf，iperf和treno提供更加丰富的feature，但ttcp更加容易找到。

通过**netstat**进行流量测量**:**

在理想的网络环境下，如果把overhead算在内，吞吐量是很接近于带宽的。但是吞吐量往往低于期望值，这种情况下，你会想要知道差异在哪。如之前所提到的，可能与硬件或软件相关。但通常是由于网络上其他数据流的影响。如果你无法确定原因，下一步就是查看你网络上的数据流。

有三种基本方法可供采用。第一，最快的方法是使用如netstat这样的工具来查看链路行为。或通过抓包来查看数据流。最后，可使用基于SNMP的工具如ntop。

要得到网络上数据流的快照，使用-i选项。举例来说：

```
bsd2# netstat -i
Name  Mtu   Network       Address            Ipkts Ierrs    Opkts Oerrs  Coll
lp0*  1500  <Link>                               0     0        0     0     0
ep0   1500  <Link>      00.60.97.06.22.22 13971293     0  1223799     1     0
ep0   1500  205.153.63    bsd2            13971293     0  1223799     1     0
tun0* 1500  <Link>                               0     0        0     0     0
sl0*  552   <Link>                               0     0        0     0     0
ppp0* 1500  <Link>                               0     0        0     0     0
lo0   16384 <Link>                             234     0      234     0     0
lo0   16384 127           localhost            234     0      234     0     0
```

输出显示了自上一次重启以来，各接口所处理的报文数量。在本例中，接口ep0收到13,971,293个没有差错(Ierrs)的报文(Ipkts)，发送了1,223,799 个报文(Opkts)，有1个差错，没有冲突（Coll）。少量错误通常并不是造成告警的原因，但各错误所占比例应当是维持在较

低水平，应该明显低于报文总量的0.1%。冲突可以稍微高一些，但应当少于数据流总量的10%。冲突数量仅包括那些影响接口的。较高数量的冲突喻示着网络负载较高，用户应当考虑分段。冲突只出现在特定媒介上。

如果你只想要单一接口的输出，可以通过-I选项指定，如：

```
bsd2# netstat -Iep0
Name  Mtu   Network       Address            Ipkts Ierrs    Opkts Oerrs  Coll
ep0   1500  <Link>      00.60.97.06.22.22 13971838     0  1223818     1     0
ep0   1500  205.153.63    bsd2            13971838     0  1223818     1     0
```

随着实现的不同，输出可能看起来有些差异，但基本信息是一样的。例如，Linux平台的输出：

```
lnx1# netstat -i
Kernel Interface table
Iface   MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   0  7366003      0      0      0    93092      0      0      0 BMRU
eth1   1500   0   289211      0      0      0    18581      0      0      0 BRU
lo     3924   0      123      0      0      0      123      0      0      0 LRU
```

如上例所示，Linux将丢失报文拆成三个目录：errors, drops,以及overruns。

不方便的是，netstat的返回值是系统自上一次重启之后的累计值。我们真正关心的是这些数值最近是怎样变化的，因为问题是在发展的，在它增长到足以显现问题之前会花费相当长的时间。

有时你会对系统做一些压力测试来看错误是否增加，可以使用ping加 –I选项或spray命令。

首先，运行netstat来得到当前值：

```
bsd2# netstat -Iep0
Name  Mtu   Network       Address            Ipkts Ierrs    Opkts Oerrs  Coll
ep0   1500  <Link>      00.60.97.06.22.22 13978296     0  1228137     1     0
ep0   1500  205.153.63    bsd2            13978296     0  1228137     1     0
```

接下来，发送大量报文到目的地址。本例中，发送了1000个UDP报文：

```
bsd1# spray -c1000 205.153.63.239
sending 1000 packets of lnth 86 to 205.153.63.239 ...
        in 0.09 seconds elapsed time
        464 packets (46.40%) dropped
Sent:   11267 packets/sec, 946.3K bytes/sec
Rcvd:   6039 packets/sec, 507.2K bytes/sec
```

注意到该测试超出了网络容量，因为464个报文被丢弃了。这可能意味着网络拥塞。更加可能的是，主机正在尝试与一个慢速设备通信。当spray在相反方向运行时，没有报文丢弃。

最后，回到netstat来看看是否存在问题：

```
bsd2# netstat -Iep0
Name  Mtu   Network       Address            Ipkts Ierrs    Opkts Oerrs  Coll
ep0   1500  <Link>      00.60.97.06.22.22 13978964     0  1228156     1     0
ep0   1500  205.153.63    bsd2            13978964     0  1228156     1     0
```

本例显示没有问题。

如果显示有问题，可以通过-s选项来得到。输出数据量可能有点吓人，但可以提供丰富的信息。信息按照协议和错误类型来分段，如bad checksum或报文头不完整。

在某些系统上，两次-s选项显示非零值的总和，如下所示：

```
bsd2# netstat -s -s
ip:
        255 total packets received
        255 packets for this host
        114 packets sent from this host
icmp:
        ICMP address mask responses are disabled
igmp:
tcp:
        107 packets sent
                81 data packets (8272 bytes)
                26 ack-only packets (25 delayed)
        140 packets received
                77 acks (for 8271 bytes)
                86 packets (153 bytes) received in-sequence
        1 connection accept
        1 connection established (including accepts)
        77 segments updated rtt (of 78 attempts)
        2 correct ACK header predictions
        62 correct data packet header predictions
udp:
        115 datagrams received
        108 broadcast/multicast datagrams dropped due to no socket
        7 delivered
        7 datagrams output
```

通过-p选项显示某一协议的汇总信息，下例显示TCP非零值的统计信息：

```
bsd2# netstat -p tcp -s -s
tcp:
        147 packets sent
                121 data packets (10513 bytes)
                26 ack-only packets (25 delayed)
        205 packets received
                116 acks (for 10512 bytes)
                122 packets (191 bytes) received in-sequence
        1 connection accept
        1 connection established (including accepts)
        116 segments updated rtt (of 117 attempts)
        2 correct ACK header predictions
        88 correct data packet header predictions
```

解释这一结果是需要一些经验的。一开始可以从大量错误信息开始看起。接下来，识别错误类型。通常，input error是由于硬件故障应期的。 Output error是由本地主机的问题造成。Data corruption,例如错误校验和，通常产生于服务器。冲突往往意味着网络拥塞。当然，这只是一般情况。

参考

Network Troubleshooting Tools

# 网络基本功（十七）：细说tcpdump的妙用（上）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese

## 介绍

tcpdump命令最初设计用于观察TCP/IP性能问题，它是一个用于截取网络分组，并输出分组内容的工具。tcpdump可以将网络中传送的数据包的报文头完全截获下来提供分析，它支持针对网络层、协议、主机、网络或端口的过滤，并提供and, or, not等逻辑语句来帮助用户去掉无用的信息。

## 更多信息

使用**tcpdump:**

Unix命令tee通常用来允许用户查看并记录Unix会话的输出。使用tcpdump结合tee加上-l选项来实现，命令格式如下：

```
bsd1# tcpdump -l  | tee outfile
```

另一种方式是通过-w选项直接将抓取数据写入文件中。之后通过tcpdump -r选项来读取。抓取数据可以输入：

```
bsd1# tcpdump -w rawfile
```

然后将raw文件转化成text文件：

```
bsd1# tcpdump -r rawfile   > textfile
```

**tcpdump**选项:

tcpdump选项可划分为四大类型：控制tcpdump程序行为，控制数据怎样显示，控制显示什么数据，以及过滤命令。

控制程序行为

这一类命令行选项影响程序行为，包括数据收集的方式。之前已介绍了两个例子：-r和-w。-w选项允许用户将输出重定向到一个文件，之后可通过-r选项将捕获数据显示出来。

如果用户知道需要捕获的报文数量或对于数量有一个上限，可使用-c选项。则当达到该数量时程序自动终止，而无需使用kill命令或Ctrl-C。下例中，收集到100个报文之后tcpdump终止：

```
bsd1# tcpdump -c100
```

如果用户在多余一个网络接口上运行tcpdump，用户可以通过-i选项指定接口。在不确定的情况下，可使用ifconfig –a来检查哪一个接口可用及对应哪一个网络。例如，一台机器有两个C级接口，xl0接口IP地址 205.153.63.238，xl1接口IP地址205.153.61.178。要捕捉205.153.61.0网络的数据流，使用以下命令：

```
bsd1# tcpdump -i xl1
```

没有指定接口时，tcpdump默认为最低编号接口。

-p选项将网卡接口设置为非混杂模式。这一选项理论上将限制为捕获接口上的正常数据流——来自或发往主机，多播数据，以及广播数据。

-s选项控制数据的截取长度。通常，tcpdump默认为一最大字节数量并只会从单一报文中截取到该数量长度。实际字节数取决于操作系统的设备驱动。通过默认值来截取合适的报文头，而舍弃不必要的报文数据。

如果用户需截取更多数据，通过-s选项来指定字节数。也可以用-s来减少截取字节数。对于少于或等于200字节的报文，以下命令会截取完整报文： bsd1# tcpdump -s200

更长的报文会被缩短为200字节。

控制信息如何显示

-a，-n，-N和-f选项决定了地址信息是如何显示的。-a选项强制将网络地址显示为名称，-n阻止将地址显示为名字，-N阻止将域名转换。-f选项阻止远端名称解析。下例中，从sloan.lander.edu (205.153.63.30) ing远程站点，分别不加选项，-a，-n，-N，-f。（选项-c1限制抓取1个报文）

```
bsd1# tcpdump -c1 host 192.31.7.130
tcpdump: listening on xl0
14:16:35.897342 sloan.lander.edu > cio-sys.cisco.com: icmp: echo request
bsd1# tcpdump -c1 -a host 192.31.7.130
tcpdump: listening on xl0
14:16:14.567917 sloan.lander.edu > cio-sys.cisco.com: icmp: echo request
bsd1# tcpdump -c1 -n host 192.31.7.130
tcpdump: listening on xl0
14:17:09.737597 205.153.63.30 > 192.31.7.130: icmp: echo request
bsd1# tcpdump -c1 -N host 192.31.7.130
tcpdump: listening on xl0
14:17:28.891045 sloan > cio-sys: icmp: echo request
bsd1# tcpdump -c1 -f host 192.31.7.130
tcpdump: listening on xl0
14:17:49.274907 sloan.lander.edu > 192.31.7.130: icmp: echo request
```

默认为-a选项。

-t和-tt选项控制时间戳的打印。-t选项不显示时间戳而-tt选项显示无格式的时间戳。以下命令显示了tcpdump命令无选项，-t选项，-tt选项的同一报文：

```
12:36:54.772066 sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647
sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647 (DF)
934303014.772066 sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647
```

控制显示什么数据

可以通过-v和-vv选项来打印更多详细信息。例如，-v选项将会打印TTL字段。要显示较少信息，使用-q，或quiet选项。一下为同一报文分别使用-q选项，无选项，-v选项，和-vv选项的输出。

```
12:36:54.772066 sloan.lander.edu.1174 &gt; 205.153.63.238.telnet: tcp 0 (DF)
12:36:54.772066 sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647
12:36:54.772066 sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647
12:36:54.772066 sloan.lander.edu.1174 > 205.153.63.238.telnet: . ack 3259091394 win 8647
```

-e选项用于显示链路层头信息。上例中-e选项的输出为：

```
12:36:54.772066 0:10:5a:a1:e9:8 0:10:5a:e3:37:c ip 60:
sloan.lander.edu.1174 &gt; 205.153.63.238.telnet: . ack 3259091394 win 8647 (DF)
```

0:10:5a:a1:e9:8是sloan.lander.edu中3Com卡的以太网地址，0:10:5a:e3:37:c是205.153.63.238中3Com卡的以太网地址。

-x选项将报文以十六进制形式dump出来，排除了链路层报文头。-x和-vv选项报文显示如下：

```
13:57:12.719718 bsd1.lander.edu.1657 > 205.153.60.5.domain: 11587+ A? www.microsoft.com.
                         4500 003f a189 0000 4011 c43a cd99 3db2
                         cd99 3c05 0679 0035 002b 06d9 2d43 0100
                         0001 0000 0000 0000 0377 7777 096d 6963
                         726f 736f 6674 0363 6f6d 0000 0100 01
```

（未完待续）

# 参考

Network Troubleshooting Tools

# 网络基本功（十八）：细说tcpdump的妙用（下）

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese




## 更多信息

（承接上文）

过滤:

要有效地使用tcpdump，掌握过滤器非常必要的。过滤允许用户指定想要抓取的数据流，从而用户可以专注于感兴趣的数据。此外，ethereal这样的工具使用tcpdump过滤语法来抓取数据流。

如果用户很清楚对何种数据流不感兴趣，可以将这部分数据排除在外。如果用户不确定需要什么数据，可以将源数据收集到文件之后在读取时应用过滤器。实际应用中，需要经常在两种方式之间转换。

简单的过滤器是加在命令行之后的关键字。但是，复杂的命令是由逻辑和关系运算符构成的。对于这样的情况，通常最好用-F选项将过滤器存储在文件中。例如，假设testfilter 是一个包含过滤主机205.153.63.30的文本文件，之后输入tcpdump –Ftestfilter等效于输入命令tcpdump host 205.153.63.30。通常，这一功能只在复杂过滤器时使用。但是，同一命令中命令行过滤器和文件过滤器不能混用。

地址过滤：

过滤器可以按照地址选择数据流。例如，考虑如下命令：

```
bsd1# tcpdump host 205.153.63.30
```

该命令抓取所有来自以及发往IP地址205.153.63.30的主机。主机可以通过名称或IP地址来选定。虽然指定的是IP地址，但抓取数据流并不限于IP数据流，实际上，过滤器也会抓到ARP数据流。限定仅抓取特定协议的数据流要求更复杂的过滤器。

有若干种方式可以指定和限制地址，下例是通过机器的以太网地址来选择数据流：

```
bsd1# tcpdump ether host 0:10:5a:e3:37:c
```

数据流可进一步限制为单向，分别用src或dst指定数据流的来源或目的地。下例显示了发送到主机205.153.63.30 的数据流：

bsd1# tcpdump dst 205.153.63.30

注意到本例中host被省略了。在某些例子中省略是没问题的，但添加这些关键字通常更安全些。

广播和多播数据相应可以使用broadcast和multicast。由于多播和广播数据流在链路层和网络层所指定的数据流是不同的，所以这两种过滤器各有两种形式。过滤器ether multicast抓取以太网多播地址的数据流，ip multicast抓取IP多播地址数据流。广播数据流也是类似的使用方法。注意多播过滤器也会抓到广播数据流。

除了抓取特定主机以外，还可以抓取特定网络。例如，以下命令限制抓取来自或发往205.153.60.0的报文：

```
bsd1# tcpdump net 205.153.60
```

以下命令也可以做同样的事情：

```
bsd1# tcpdump net 205.153.60.0 mask 255.255.255.0
```

而以下命令由于最后的.0就无法正常工作：

```
bsd1# tcpdump net 205.153.60.0
```

协议及端口过滤：

限制抓取指定协议如IP，Appletalk或TCP。还可以限制建立在这些协议之上的服务，如DNS或RIP。这类抓取可以通过三种方式进行：使用tcpdump关键字，通过协议关键字proto，或通过服务使用port关键字。

一些协议名能够被tcpdump识别到因此可通过关键字来指定。以下命令限制抓取IP数据流：

```
bsd1# tcpdump ip
```

当然，IP数据流包括TCP数据流，UDP数据流，等等。

如果仅抓取TCP数据流，可以使用：

```
bsd1# tcpdump tcp
```

tcpdump可识别的关键字包括ip, igmp, tcp, udp, and icmp。

有很多传输层服务没有可以识别的关键字。在这种情况下，可以使用关键字proto或ip proto加上/etc/protocols能够找到的协议名或相应的协议编号。例如，以下两种方式都会查找OSPF报文：

```
bsd1# tcpdump ip proto ospf
bsd1# tcpdump ip proto 89
```

内嵌的关键字可能会造成问题。下面的例子中，无法使用tcp关键字，或必须使用数字。例如，下面的例子是正常工作的：

```
bsd#1 tcpdump ip proto 6
```

另一方面，不能使用proto加上tcp:

```
bsd#1 tcpdump ip proto tcp
```

会产生问题。

对于更高层级的建立于底层协议之上的服务，必须使用关键字port。以下两者会采集DNS数据流：

```
bsd#1 tcpdump port domain
bds#1 tcpdump port 53
```

第一条命令中，关键字domain能够通过查找/etc/services来解析。在传输层协议有歧义的情况下，可以将端口限制为指定协议。考虑如下命令：

```
bsd#1 tcpdump udp port domain
```

这会抓取使用UDP的DNS名查找但不包括使用TCP的DNS zone传输数据。而之前的两条命令会同时抓取这两种数据。

报文特征：

过滤器也可以基于报文特征比如报文长度或特定字段的内容，过滤器必须包含关系运算符。要指定长度，使用关键字less或greater。如下例所示：

```
bsd1# tcpdump greater 200
```

该命令收集长度大于200字节的报文。

根据报文内容过滤更加复杂，因为用户必须理解报文头的结构。但是尽管如此，或者说正因如此，这一方式能够使用户最大限度的控制抓取的数据。

一般使用语法 proto [ expr : size ]。字段proto指定要查看的报文头——ip则查看IP头，tcp则查看TCP头，以此类推。expr字段给出从报文头索引0开始的位移。即：报文头的第一个字节为0，第二字节为1，以此类推。size字段是可选的，指定需要使用的字节数，1，2或4。

```
bsd1# tcpdump "ip[9] = 6"
```

查看第十字节的IP头，协议值为6。注意这里必须使用引号。撇号或引号都可以，但反引号将无法正常工作。

```
bsd1# tcpdump tcp
```

也是等效的，因为TCP协议编号为6。

这一方式常常作为掩码来选择特定比特位。值可以是十六进制。可通过语法&加上比特掩码来指定。下例提取从以太网头第一字节开始（即目的地址第一字节），提取低阶比特位，并确保该位不为0：

```
bsd1# tcpdump 'ether[0] & 1 != 0'
```

该条件会选取广播和多播报文。

以上两个例子都有更好的方法来匹配报文。作为一个更实际的例子，考虑以下命令：

```
bsd1# tcpdump "tcp[13] & 0x03 != 0"
```

该过滤器跳过TCP头的13个字节，提取flag字节。掩码0x03选择第一和第二比特位，即FIN和SYN位。如果其中一位不为0则报文被抓取。此命令会抓取TCP连接建立及关闭报文。

不要将逻辑运算符与关系运算符混淆。比如想tcp src port > 23这样的表达式就无法正常工作。因为tcp src port表达式返回值为true或false，而不是一个数值，所以无法与数值进行比较。如果需要查找端口号大于23的所有TCP数据流，必须从报文头提取端口字段，使用表达式“tcp[0:2] & 0xffff > 0x0017”。

# 网络基本功（十九）：细说NAT原理与配置

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

NAT技术让少数公有IP地址被使用私有地址的大量主机所共享。这一机制允许远多于IP地址空间所支持的主机共享网络。同时，由于NAT屏蔽了内部网络，也为局域网内的机器提供了安全保障。

NAT的基本实施过程包括使用一个预留给本地IP网络的私有地址成立组织的内部网络，同时分配给组织一个或多个公网IP地址，并在本地网络与公网之间安装一个或多个具有NAT功能的路由器。NAT路由器实现的功能包括将数据报中私网地址转换成公网地址，反向亦然。当有报文通过时，网络地址转换其不仅检查报文信息，还将报文头中的IP地址和端口信息进行修改，以使处于NAT之后的机器共享少数公网IP地址。

## 更多信息

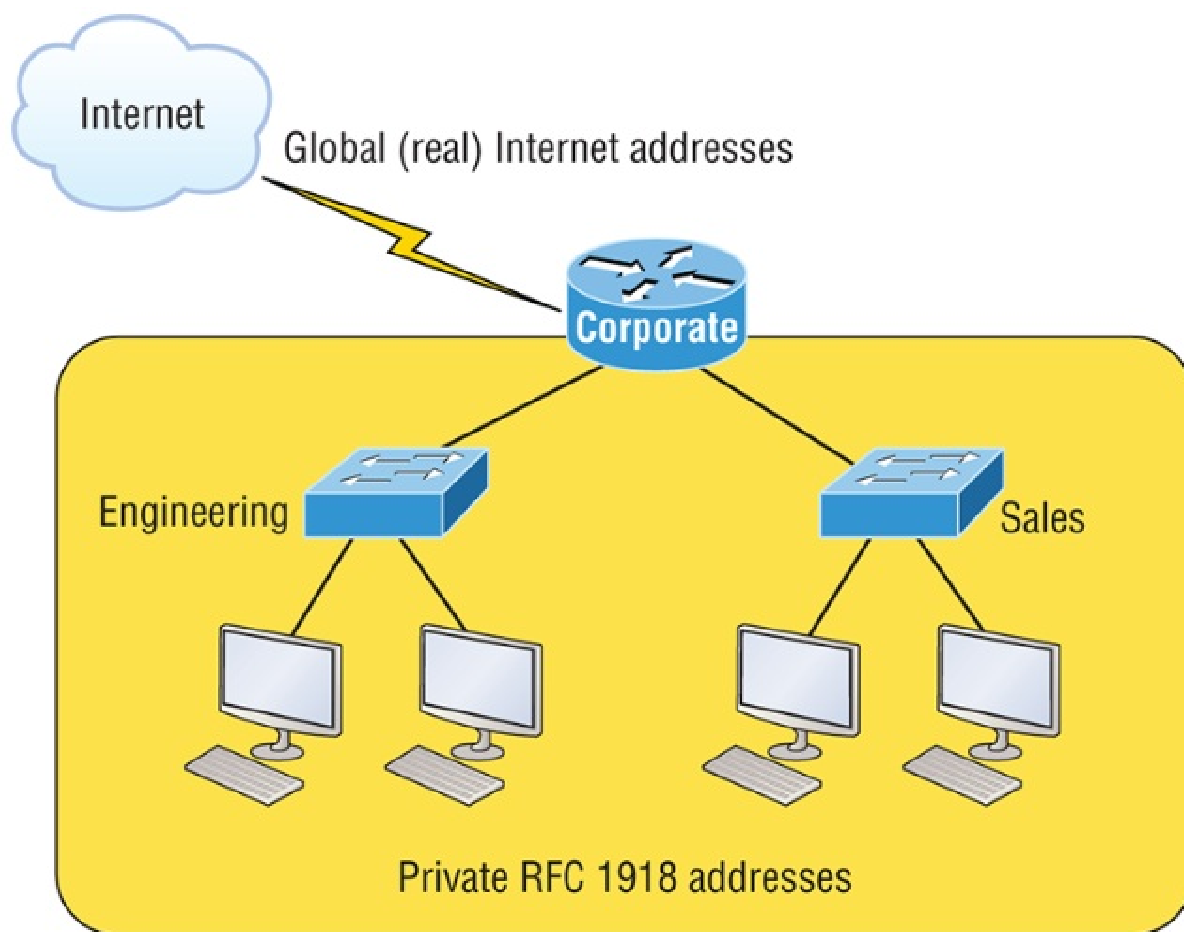
何时使用**NAT?**

因为NAT能够减少在网络环境中所需的公共IP地址需求，因此当两家公司重复内部地址合并时，这一技术是很有帮助的。当组织改变其Internet服务供应商（ISP），但网络管理员不想改变内部地址方案时，NAT也是一个很好用的工具。

以下是应用NAT的场景：

- 用户需要访问Internet但主机没有全球唯一的IP地址
- 用户更改ISP需要对网络重新编号
- 用户需要合并地址重复的内网

通常NAT应用于边界路由器。例如，下图中NAT应用于企业连接到Internet的路由器上：




**NAT**的优势与不足：

| 优势 | 不足 |
|---|---|
| 节约合法注册地址 | 转换导致交换路径延时 |
| 解决地址重叠问题 | 导致端到端IP地址无法追溯 |
| 提高访问Internet灵活性 | 某些应用程序无法使用 |
| 网络变动无需地址重新编号 | |

网络地址转换类型：

静态NAT：此类NAT在本地和全局地址之间做一到一的永久映射。须注意静态NAT要求用户对每一台主机都有一个真实的Internet IP地址。

动态NAT：允许用户将一个未登记的IP地址映射到一个登记的IP地址池中的一个。采用动态分配的方法将外部合法地址映射到内部网络，无需像静态NAT那样，通过对路由器进行静态配置来将内部地址映射到外部地址，但是必须有足够的真正的IP地址来进行收发包。

端口NAT（PAT）：最为流行的NAT配置类型。通过多个源端口，将多个未登记的IP地址映射到一个合法IP地址（多到一）。使用PAT能够使上千个用户仅使用一个全局IP地址连接到Internet。

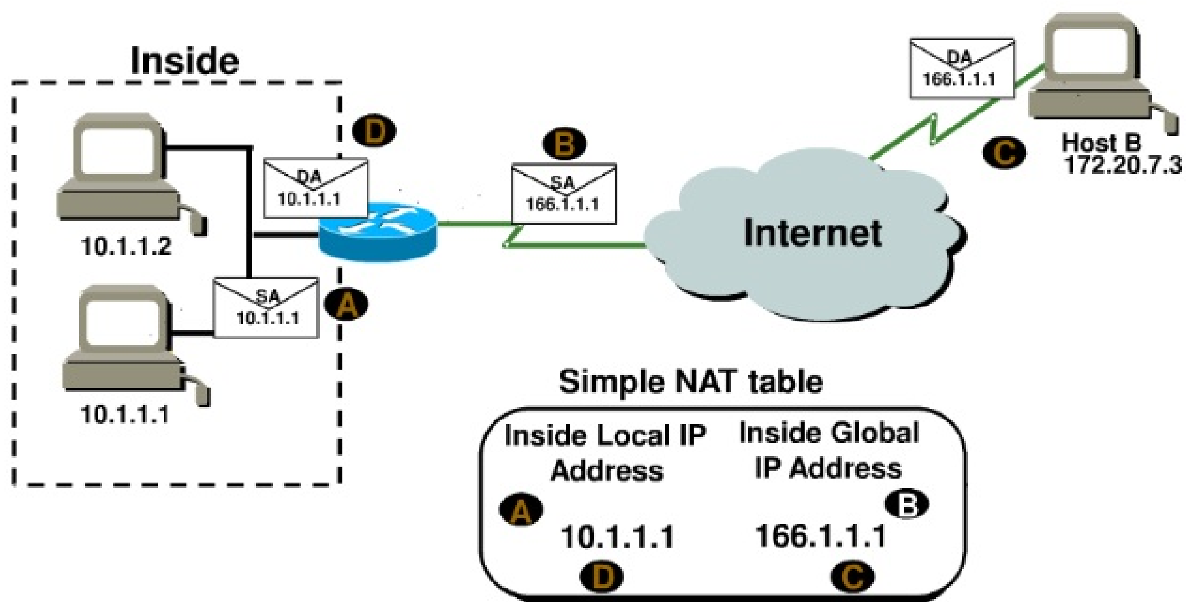
NAT术语：

NAT术语还是比较直观的。NAT地址转换之后成为全局地址。通常是Internet上使用的公网地址。如果不访问Internet的话就不需要用到。

本地地址：NAT地址转换之前用到的地址。内部本地地址实际上是尝试访问Internet的发送主机的私有地址。外部本地地址通常是连接到用户ISP的路由器接口，也是报文开始传输的公有地址。

转换之后，内部本地地址之后被称为内部全局地址，而外部全局地址成为目标主机的地址。如下表所示：

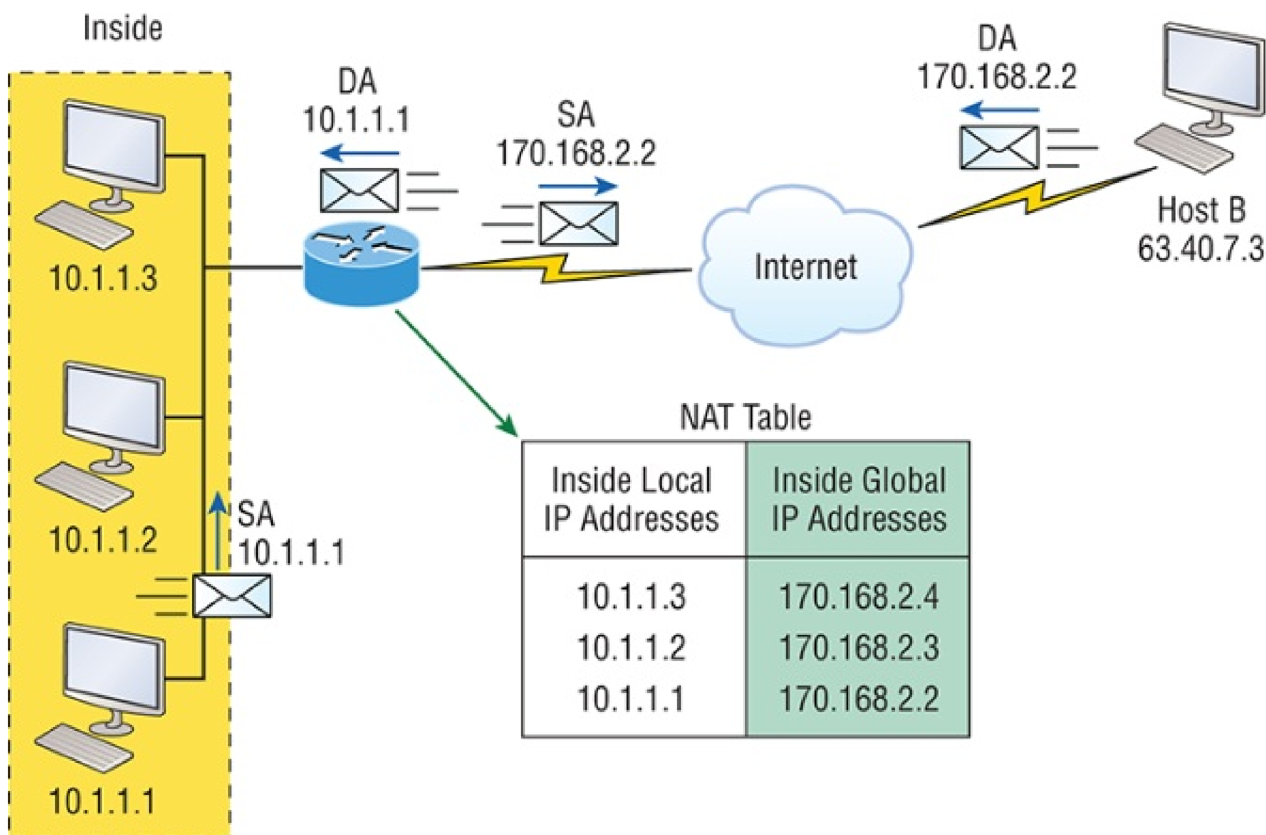
| 名称 | 含义 |
|---|---|
| 内部本地 | 转换前的源主机内部地址 |
| 外部本地 | Internet上识别到源主机的地址。通常是连接到ISP的路由器接口——真实的Internet地址。 |
| 内部全局 | 转换后连接到Internet的源主机地址。也是真实的Internet地址 |
| 外部全局 | 外部目标主机地址，同样是真实的Internet地址 |




NAT实现细节：

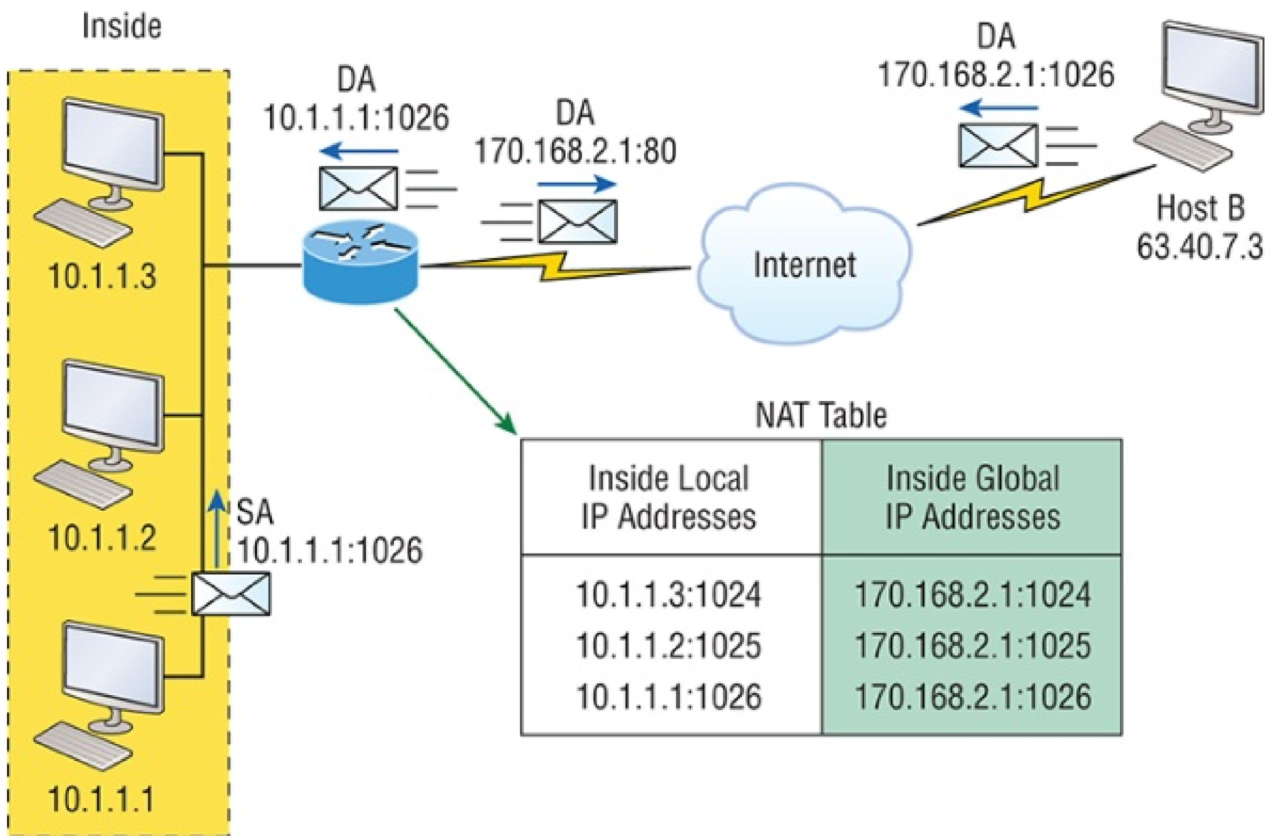
下图中，主机10.1.1.1将报文发送到有NAT功能的边界路由器。路由器将源IP地址识别为内部本地IP地址，在报文中转换源IP地址，并在NAT表中记录此次转换。

配有新转换源地址的报文发送到外部接口。外部主机将报文发送给目的主机并且NAT路由器通过NAT表将内部全局IP地址转换回内部本地IP地址。




PAT方式中，所有内部主机都转换为一个IP地址。如下图所示，除了内部本地IP地址和内部全局IP地址以外，还多了一个端口号。端口号帮助路由器识别哪一台主机应当收到返回数据。路由器使用来自各主机的源端口好来区别他们各自发出的数据。注意当报文离开路由器时有一个目标端口号80，而HTTP服务器将报文发回时目的端口号为1026。从而允许NAT转换路由器区别NAT表中的主机然后将目的IP地址转换回内部本地地址。

本例中，端口号在传输层用户识别本地主机。如果必须要使用真实全局IP地址来识别源主机，那就只能通过静态NAT，并且会用光所有地址。PAT允许我们在传输层识别主机，从而理论上一个真实IP地址可被65，000台主机共享。




静态**NAT**配置：

```
ip nat inside source static 10.1.1.1 170.46.2.2
!
interface Ethernet0
ip address 10.1.1.10 255.255.255.0
ip nat inside
!
interface Serial0
ip address 170.46.2.1 255.255.255.0
ip nat outside
!
```

在第一个路由器输出中， ip nat inside source 命令指定需要转换的IP地址。本例中，此命令配置了内部本地IP地址10.1.1.1到外部全局IP地址170.46.2.2的静态配置。

在各接口下都有一条ip nat命令。ip nat inside命令将该接口识别为内部接口，ip nat outside命令将该接口识别为外部接口。回头看 ip nat inside source 命令，该命令将内部接口作为转换的源或起点。也可以这样使用：ip nat outside source。该选项表明指定的外部接口会成为转换的源或起点。

动态**NAT**配置：

动态NAT表示将一个地址池当作真实IP地址提供给内部一组用户。由于不使用端口号，对于同时尝试访问外部网络的用户必须提供真实的IP地址。

以下是动态NAT配置的示例输出：

```
ip nat pool todd 170.168.2.3 170.168.2.254
    netmask 255.255.255.0
ip nat inside source list 1 pool todd
!
interface Ethernet0
ip address 10.1.1.10 255.255.255.0
ip nat inside
!
interface Serial0
ip address 170.168.2.1 255.255.255.0
ip nat outside
!
access-list 1 permit 10.1.1.0 0.0.0.255
!
```

ip nat inside source list 1 pool todd 命令告知路由器将匹配access-list 1的IP地址转换到名为todd的IP NAT池中的一个地址。这里ACL并不是出于安全因素通过允许或拒绝数据来过滤报文。本例中，它是用来选择或指定我们感兴趣的数据流。当数据流与接入列表相匹配，就被拉入NAT进程转换。

命令 ip nat pool todd 170.168.2.3 192.168.2.254 netmask 255.255.255.0用来创建地址池，之后被分配给请求全局地址的主机。做Cisco NAT故障排查时，一定要检查池中确保有足够地址提供转换给内部主机。最后，确保池名匹配，注意区分大小写。

端口**NAT**配置：

以下是端口NAT配置的示例输出：

```
ip nat pool globalnet 170.168.2.1 170.168.2.1 netmask 255.255.255.0
ip nat inside source list 1 pool globalnet overload
!
interface Ethernet0/0
ip address 10.1.1.10 255.255.255.0
ip nat inside
!
interface Serial0/0
ip address 170.168.2.1 255.255.255.0
ip nat outside
!
access-list 1 permit 10.1.1.0 0.0.0.255
```

端口NAT与动态NAT配置的不同之处在于：

地址池变为只有一个IP地址

在ip nat inside source命令最后加入overload关键字。

本例中一个关键元素是使用了池中的一个IP地址作为外部接口IP地址。如果有其他可用地址如170.168.2.2可作为额外地址，这样做在内部大量用户同时为活跃状态，需要不止一个重载IP地址时很有帮助。

# 参考

CCENT

# 网络基本功（二十）：细说ICMP和ARP

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese

## 介绍

ICMP是网络控制消息协议，主要用于传递查询报文与差错报文。ARP是地址解析协议，它的作用是在以太网环境下，通过3层的IP地址来找寻2层的MAC地址，得到一张ARP缓存表。转发数据的时候根据ARP缓存表来进行传输。

## 更多信息

**IMCP:**

Internet操作是由路由器严密监控的。当路由器端处理报文时如有意外发生，事件通过ICMP报告给发送端。ICMP也用来测试Internet。ICMP信息封装在IP报文中，最重要的一部分如下表所列：

| Message type | Description |
|---|---|
| Destination unreachable | Packet could not be delivered |
| Time exceeded | Time to live field hit 0 |
| Parameter problem | Invalid header field |
| Source quench | Choke packet |
| Redirect | Teach a router about geography |
| Echo and echo reply | Check if a machine is alive |
| Timestamp request/reply | Same as Echo, but with timestamp |
| Router advertisement/solicitation | Find a nearby router |

DESTINATION UNREACHABLE消息用于当路由器无法找到目标地址或当设置了DF位的报文无法递送，因为路径上存在“小报文”网络。

TIME EXCEEDED消息是由于报文TTL（Time to live）计数器到达0时。该事件是报文在回环，或计数器值设置过低的迹象。对于这一错误信息的聪明的应用是traceroute工具，traceroute发现从主机到目的IP地址路径上的路由器。它向目的地发送IP包，第一次的时候，将TTL设置为1，引发第一个路由器的Time Exceeded错误。这样，第一个路由器回复ICMP包，从而让出发主机知道途径的第一个路由器的信息。随后TTL被设置为2、3、4，...，直到

到达目的主机。这样，沿途的每个路由器都会向出发主机发送ICMP包来汇报错误。traceroute将ICMP包的信息打印在屏幕上，就是接力路径的信息了。这并不是TIME EXCEEDED信息的本意，但却是非常有用的故障排查工具。

PARAMETER PROBLEM信息表示报文头字段发现了非法值。这一问题表明发送主机的IP软件或可能是途经的路由器发生了bug。

SOURCE QUENCH信息以前用来节制发送太多报文的主机。当主机接收到该信息，它预计将放缓发送报文。现在很少使用，因为当拥塞发生时，这类报文会起到火上浇油的作用，而且也不清楚如何做出回应。Internet中的拥塞控制现在大部分在传输层完成，使用报文丢失作为拥塞信号。

REDIRECT信息用于路由器发现报文被错误路由的时候。路由器用该信息告知发送主机更新合适的路径。

主机发送ECHO和ECHO REPLY信息以查看目前的目的地址是否可到达或是否alive。接收到ECHO信息之后，目的地址预计会发回一条ECHO REPLY信息。这些信息用在ping工具中来查看主机是否up以及是否挂在网上。

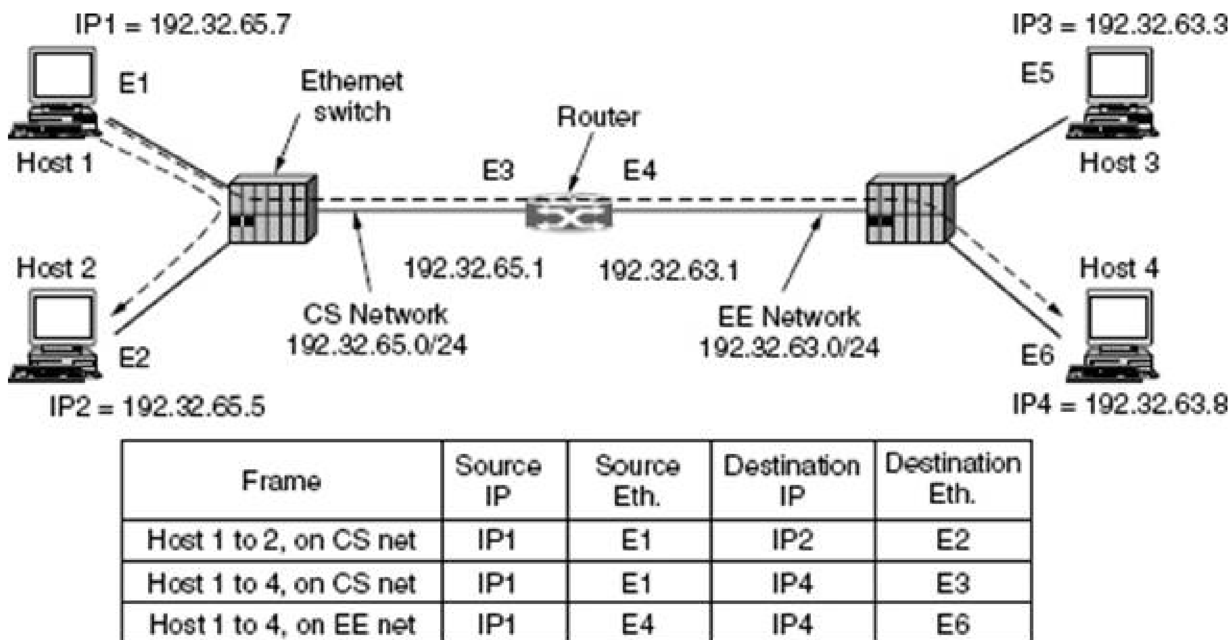
TIMESTAMP REQUEST和TIME REPLY信息是类似的，除了信息的到达时间以及回复的离开时间是记录在回复中的。这一工具可用于衡量网络性能。

ROUTER ADVERTISEMENT和ROUTER SOLICITATION信息用于主机发现附近的路由器。主机需要从至少一个路由器学习IP地址来发送报文。

**ARP:**

尽管Internet上的每台设备都有一个或多个IP地址，仅用这些地址仍然不能发送报文。数据链路层网卡如以太网卡不理解Internet地址。对于以太网，每一个以太网卡都有一个48bit的以太网地址。网卡基于这48bit以太网地址收发帧，网卡与32bit IP地址没有关系。

从而产生一个问题：IP地址是如何映射到数据链路层地址，如以太网地址的呢？解释这一问题，让我们以下图为例：一个小型校园安装了两个/24网络。其中一个（CS）是交换以太网，位于Computer Science部门。网络地址为192.32.65.0/24。另一个局域网（EE）也是一个交换以太网，位于Electrical Engineering部门网络地址为192.32.63.0/24。这两个局域网通过IP路由器互连。以太网上的各台设备以及路由器各接口都有唯一的以太网地址，标签从E1到E6，以及CS或EE网络上唯一的IP地址。




| Frame | Source IP | Source Eth. | Destination IP | Destination Eth. |
|---|---|---|---|---|
| Host 1 to 2, on CS net | IP1 | E1 | IP2 | E2 |
| Host 1 to 4, on CS net | IP1 | E1 | IP4 | E3 |
| Host 1 to 4, on EE net | IP1 | E4 | IP4 | E6 |

我们首先看一下host 1的用户如何向CS上的host 2用户发送报文。首先假设发送方知道目标接收方的名字，如xx.cs.uni.edu。第一步是查找host 2的IP地址。这一查找是通过DNS来完成的，DNS之后返回host 2的IP地址（192.32.65.5）。

host 1的上层软件将目标地址192.32.65.5植入报文中并交给IP软件发送。IP软件查看该地址发现目标地址在CS网络上（即本地网络）。但是，还需要查找目的以太网地址来发送帧。一种解决方式是通过系统配置文件来将IP地址映射到以太网地址。当然这种方式是可能的，但对于有上千台设备的大型企业来说要保证这些文件都是更新状态是一项耗时的工作。

比较好的方式是host 1发送一个广播报文到以太网询问谁有IP地址192.32.65.5。广播报文到达每一个CS网上的设备，各台设备检查自己的IP地址。只有host 2会回复自己的以太网地址E2。通过这种方式host 1学习到IP地址192.32.65.5的以太网地址E2。这种提问和回复的协议就称为ARP(Address Resolution Protocol)。使用ARP的一个优势是它的简单性。系统管理员无需指定各台设备的IP地址以及子网掩码，ARP自动完成剩下的工作。

此时，host 1上面的IP软件构造以太网地址E2的报文，将IP报文（目的地址192.32.65.5）放在载荷部分。host 2的以太网卡检测到该帧，识别目标地址是自己，把它捞出来，产生一个中断。以太网驱动从载荷中将IP报文提取出来并传递给IP软件，软件查看到此报文地址正确并予以处理。

提高ARP效率有很多种优化方法。运行ARP的设备将其结果放入缓存之中，以备短期内需要再次连接同一台设备。下一次可在设备的缓存中找到映射结果，就无需第二次广播。很多情况下，host 2需要发送一个回复，迫使它运行ARP来确定发送方的以太网地址。在host 1的ARP报文中包含IP到Ethernet映射可避免这一ARP广播。当ARP广播到达host 2，连接对（192.32.65.7, E1）进入host 2的ARP缓存。实际上，以太网上的所有设备都可以将这一映射放入自己的ARP缓存中。

为了让映射能够更改，例如，当配置一台主机使用心得IP地址（但保留旧的以太网地址），几秒钟过后ARP缓存中的表项会过期。为了保持缓存信息更新并且优化性能，比较好的方法是每一台设备在配置时都广播它的映射信息。广播通常以ARP查找自己的IP地址的方式来完成。应当不会收到响应，但该广播的副作用是使每一台设备的ARP缓存都得到更新。这称为免费ARP（gratuitous ARP）。如果收到回复（不期望地），则两台设备指定了同一IP地址。网络管理员需解决这一问题才能使两台设备共同使用网络。

再看上图，假设host 1想要向网络EE上的host 4(192.32.63.8)发送报文。host 1会发现目标IP地址不在CS网络上，它会将所有这类远端网络数据流发送给路由器，也称为默认网关（default gateway）。习惯上，默认网关是网络上的最低地址（198.31.65.1）。要发送帧给路由器，host 1还是必须知道路由器接口在CS网络上的以太网地址。路由器通过发送198.31.65.1的ARP广播来学习到E3地址，然后发送帧。路由器在Internet路径上将报文从一个路由器发送到下一个使用相同的查找机制。

路由器的以太网卡收到此帧后将报文发给IP软件。从网络掩码中得知该报文应当发送到EE。如果路由器不知道host 4的以太网地址，就会再次使用ARP。上图列出了在CS和EE子网上观察到的帧中出现的源和目的以太网及IP地址。发现到各子网中以太网地址改变而IP地址保持不变（因为IP地址指明跨越所有互连子网的终点）。

从host 1发送报文到host 4，而host 1不知道host 4位于不同网络也是可能的。解决方法是让CS子网上的路由器回复查找host 4的ARP并将自己的以太网地址E3作为回复内容。由于host 4无法看到ARP请求（路由器不会转发以太网广播）所以无法直接回复。路由器之后会接收发往192.32.63.8的帧并转发到EE子网。这一方式称为代理ARP（proxy ARP）。用在一台主机想要出现在一个子网上但实际上位于另一子网的特定情形。

# 参考

Computer Networks

# 网络基本功（二十一）：细说HTTP（上）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese



## 介绍

HTTP是一个由请求与响应组成的客户端与服务端交互协议。浏览器发送一个HTTP请求到指定的URL地址，持有此URL地址的WEB服务器将返回一个HTTP请求。请求的类型有GET, POST, HEAD, PUT, DELETE, OPTIONS和TRACE等。

## 更多信息

**HTTP**操作模式与客户端**/**服务器通信**:**

HTTP只关心一个功能：从web服务器到web客户端的超文本文件以及其他文件的传输。从通信的角度来看，客户端主要负责发送请求给服务器，服务器对请求作出响应。相比FTP和SMTP这样需要多个通信步骤和命令/响应序列的应用层协议，HTTP更像BOOTP和ARP。

基本的**HTTP**客户端**/**服务器通信：

最简单的HTTP操作包括一个使用web浏览器的HTTP客户端，和一个HTTP服务器，通常称为web服务器。在TCP连接创建之后，以下两步通信过程如下：

客户端请求：HTTP客户端根据HTTP协议标准发送HTTP请求信息，该信息指定客户端想要获取的资源或包括准备提供给服务器的信息。

服务器响应：服务器读取并解释该请求。对请求作出相应行为并创建HTTP响应信息，发回给客户端。响应信息包括该请求是否成功，也包括客户端请求的资源内容。

**HTTP**消息格式：

使用HTTP的设备通信都是通过HTTP消息来完成，其中只有两种类型：请求和响应。客户端通常发送请求和接收响应，服务器接收请求和发送响应。信息使用的是文本的形式。

常规HTTP消息格式如下所示：

```
<起始行>
<首部字段>
<空白行>
[<主体>]
[<尾部>]
```

起始行包含消息的类型。请求消息中，这一行以方式的形式表明消息为请求类型，并制定一个URI(Uniform Resource Identifier)指明请求的对象资源。响应通过起始行来表明请求响应的状态信息。

首部字段HTTP定义了多种类型的首部字段。通过功能分组，除了主机头以外，几乎所有首部字段都是可选的。格式如下：<header-name>:<header-value>。

主体也是可选的，包含客户端和服务器通信所需的一系列信息，如响应的详细错误消息。更加常见的是承载文件或其他资源，HTTP标准中称为实体。由于大多数客户端请求服务器发送文件或其他资源，实体在响应信息中最为常见。

尾部，HTTP/1.1默认使用永久链接，消息在服务器与客户端之间以流的形式传输，需要标记消息的结束点和开始点。

**HTTP**请求消息：

客户端通过打开一个TCP连接发起与服务器的HTTP会话，之后发送HTTP请求信息



起始行

主要有三个用途：

- 表明客户端想要进行的命令或行为
- 指定行为想要获取的资源
- 告知服务器客户端使用的HTTP版本

起始行的语法为：

```
<METHOD><request-uri><HTTP-VERSION>
```

**Method**

method就是客户端想要服务器做什么，三种比较常用：GET，HEAD和POST。

| | |
|---|---|
| GET | 从服务器向客户端发送发送命名资源 |
| PUT | 将来自客户端的数据存储到一个命名的服务器资源中去 |
| DELETE | 从服务器中删除命名资源 |
| POST | 将客户端数据发送到一个服务器网关应用程序 |
| HEAD | 仅发送命名资源响应中的HTTP首部 |

**Request URI**

Request URI是请求所申请资源的URI。目前URI通常值符合Web URL语法的HTTP URL。有趣的是，HTTP起始行所使用的URL形式通常与HTML文件或用户输入的不同。这是因为一个完整URL中的部分信息是用来控制HTTP本身的。这是用户和HTTP客户端通信所需，而不包括在客户端对服务器的请求中。在请求中指定资源的标准方式是在起始行中加入路径和文件名（以及可选的查询信息），同时在主机头字段指定主机。

例如：假设用户输入URL：http://www.myfavoritewebsite.com:8080/chatware/chatroom.php，我们不需要发送http:到服务器。客户端将余下的信息拆分成URI /chatware/chatroom.php主机行会包括www.myfavoritewebsite.com:8080。因此，请求的开始内容如下：

```
GET /chatware/chatroom.php HTTP/1.1
Host: www.myfavoritewebsite.com:8080
```

这一准则的例外是当请求对象是代理服务器时。这时请求就要使用完整URL的形式，以使代理可以作为初始客户端来处理该请求。请求如下所示：

```
GET http://www.myfavoritewebsite.com:8080/chatware/chatroom.php HTTP/1.1
```

请求首部

在请求首部，提供给服务器关于请求的详细信息。所有请求首部都使用相同的结构，但按照以下功能分类：

普通报头普通报头通常指消息本身，通常用于控制其处理过程或提供给接收方额外信息。这类报头不限于请求或响应信息，所以两者都可能出现。同样，也与所承载的实体没有特别关系。

请求报头 这类报头告知服务器关于客户端请求的更多信息，给予客户端更多关于请求处理的控制。例如，一些请求报头用于指定条件请求，只有在特定条件时才执行。其他告诉服务器响应信息中客户端能够徐立的格式或编码。如：

Accept 告诉服务器端，接受哪些类型的信息。

Accept-Encoding 可接受的内容编码。

Accept-Lanague 指定一种自然语言。

Connection 表示是否需要持久连接。如果Servlet看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时显著地减少下载所需要的时间。

Cookie 最重要的请求头信息之一， 每次请求时都会携带上Cookie以方便服务器端识别是否是同一个客户端。

Host host请求报头域主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来。

User-Agent用户代理，一般情况是浏览器。我们上网登陆论坛的时候，往往会看到一些欢迎信息，其中列出了客户端操作系统的名称和版本，所使用的浏览器的名称和版本，实际上，服务器应用程序就是从User-Agent这个请求报头域中获取到这些信息。User-Agent请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。

# 参考

TCP/IP Guide

# 网络基本功（二十二）：细说HTTP（下）

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

本文承接上文。

## 更多信息

**HTTP**回复信息**:**

每一个HTTP客户端发送给服务器请求都会要求服务器发回响应信息。在特定情况下，服务器会发回两条响应，一条初步响应和一条实际上的响应。一般，一个请求产生一个响应，表明服务器对于该请求的处理结果，并且响应往往消息主体还携带一个实体（文件或资源）。（微信号：EMC_Support）

响应消息格式如下：

```
<状态行>
<响应首部>
<响应实体>
```

如下图所示。

```
HTTP/1.1 200 OK

Date: Fri, 19 May 2006 21:12:58 GMT
Connection: close

Server: Apache/1.3.27
Accept-Ranges: bytes

Content-Type: text/html
Content-Length: 170
Last-Modified: Wed, 17 May 2006 10:14:49 GMT

<html>
<head>
<title>Welcome to the Amazing Site!</title>
</head>
<body>
<p>This site is under construction. Please come
back later. Sorry!</p>
</body>
</html>
```

Status Line
General Headers
Response Headers
Entity Headers
Message Body
HTTP Response

状态行

状态行是响应信息的起始行，作用有两个：告知客户端服务器使用的协议版本以及沟通客户端请求的处理结果。状态行语法格式如下：

```
<HTTP-VERSION><status-code><reason-phrase>
```

**HTTP**版本

状态行中的HTTP-VERSION标签与请求信息中的目的一样。服务器要求返回的版本号不得高于客户端发送的版本号。

响应码和文本描述

状态码和文本描述提供客户端请求处理结果的信息。服务器通过3位数字状态码告知客户端处理结果。目的是为了方便客户端HTTP软件采取合适的行动。文本描述将服务器响应显示给客户端用户。

状态代码由 3 位数字组成， 表示请求是否被理解或被满足，状态描述给出了关于状态码的简短的文字描述。状态码的第一个数字定义了响应类别，后面两位数字没有具体分类。第一个数字有5 种取值，如下所示。

- 1xx：指示信息——表示请求已经接受，继续处理
- 2xx：成功——表示请求已经被成功接收、理解、接受。
- 3xx：重定向——要完成请求必须进行更进一步的操作
- 4xx：客户端错误——请求有语法错误或请求无法实现
- 5xx：服务器端错误——服务器未能实现合法的请求。

常见状态代码、状态描述、说明：

```
200 OK      //客户端请求成功
400 Bad Request  //客户端请求有语法错误，不能被服务器所理解
401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用
403 Forbidden  //服务器收到请求，但是拒绝提供服务
404 Not Found  //请求资源不存在，eg：输入了错误的URL
500 Internal Server Error //服务器发生不可预期的错误
503 Server Unavailable  //服务器当前不能处理客户端的请求，一段时间后可能恢复正常
```

响应首部

响应首部可能包括：

Location（重定向）

Location响应报头域用于重定向接受者到一个新的位置。例如：客户端所请求的页面已不存在原先的位置，为了让客户端重定向到这个页面新的位置，服务器端可以发回Location响应报头后使用重定向语句，让客户端去访问新的域名所对应的服务器上的资源。当我们在JSP中使用重定向语句的时候，服务器端向客户端发回的响应报头中，就会有Location响应报头域。

Server响应头　　Server响应头包含处理请求的原始服务器的软件信息。此域能包含多个产品标识和注释，产品标识一般按照重要性排序。它和User-Agent请求报头域是相对应的，前者发送服务器端软件的信息，后者发送客户端软件(浏览器)和操作系统的信息。下面是Server

响应报头域的一个例子：Server: Apache-Coyote/1.1

实体头　　请求消息和响应消息都可以包含实体信息，实体信息一般由实体头域和实体组成。实体头域包含关于实体的原信息，实体头包括Allow、Content- Base、Content-Encoding、Content-Language、Content-Length、Content-Location、Content-MD5、Content-Range、Content-Type、 Etag、Expires、Last-Modified、extension-header。extension-header允许客户端定义新的实体头，但是这些域可能无法未接受方识别。实体可以是一个经过编码的字节流，它的编码方式由Content-Encoding或Content-Type定义，它的长度由Content-Length或Content-Range定义。

Content-Type实体头用于向接收方指示实体的介质类型，指定HEAD方法送到接收方的实体介质类型，或GET方法发送的请求介质类型，如："application/octet-stream"。

Last-modified：实体头指定服务器上保存内容的最后修订时间。

Accept-Ranges：这个字段说明Web服务器是否支持Range（是否支持断点续传功能），如果支持，则返回Accept-Ranges： bytes，如果不支持，则返回Accept-Ranges： none。

Content-Encoding：文档的编码（Encode）方法。它的值指示了已经被应用到实体正文的附加内容编码，因而要获得Content- Type报头域中所引用的媒体类型，必须采用相应的解码机制。Content-Encoding主要用语记录文档的压缩方法，下面是它的一个例子： Content-Encoding: gzip。如果一个实体正文采用了编码方式存储，在使用之前就必须进行解码。

Expires： 给出响应过期的日期和时间。通常，代理服务器或浏览器会缓存一些页面。当用户再次访问这些页面时，直接从缓存中加载并显示给用户，这样缩短了响应的时间，减少服务器的负载。为了让代理服务器或浏览器在一段时间后更新页面，我们可以使用Expires实体报头域指定页面过期的时间。当用户又一次访问页面时，如果Expires报头域给出的日期和时间比Date普通报头域给出的日期和时间要早(或相同)，那么代理服务器或浏览器就不会再使用缓存的页面而是从服务器上请求更新的页面。不过要注意，即使页面过期了，也并不意味着服务器上的原始资源在此时间之前或之后发生了改变。

Refresh：表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。 注意这种功能通常是通过设置HTML页面HEAD区的＜META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://host/path"＞实现。

Allow：服务器支持哪些请求方法（如GET、POST等）。

Content-Disposition：打开一个网页时，浏览器会首先看是否有Content-Disposition: attachment这一项，当是“Content-Disposition: attachment”时是下载，“Content-Disposition:inline”是在线打开文件

下面是一个响应消息

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

**HTTP方法:**

**GET**

GET方法请求服务器检索由该HTTP请求中的URL指定的资源并在回复中发给客户端。这是最基本的请求类型，也是占大多数的HTTP数据流。当你输入一个常规URL或点击一个文档中的链接，通常就是提示Web浏览器发送GET请求。

对于GET的处理取决于若干因素。如果URL正确并且服务器能够找到资源，会发送合适的响应给客户端。返回资源需取决于请求对象的特性。如果无法妥当处理请求，则会产生一个错误信息。在使用缓存的情况下，代理服务器甚至客户端自己就可以满足请求。对于某种特定报头如 If-Modified-Since 或 If-Match， GET请求的含义可能随之而改变，要求服务器仅在满足特定条件时发送资源。这类请求称为条件GET。类似的，客户端可以使用Range头来要求服务器仅发送部分资源。这类请求称为部分GET。

**HEAD**

HEAD方法同GET，但告知服务器不要发送消息实体。客户端通常使用这种方法来检查资源是否存在，状态，或文件大小，再决定是否需要服务器发送整个文件。HEAD请求的处理与GET相同，除了只返回头部而不返回实际的资源之外。

**POST**

POST方法允许客户端发送任意数据的实体到服务器以进行处理。它通常同于客户端提交例如交互式HTML信息给服务器程序，之后服务器作出行动并发回响应。这种方法用于各种在线进程。请求中的URL指定服务器上接受数据的程序名。

**PUT**

这种方法请求服务器将请求中的实体保存在请求中的URL里。PUT中，URI指明请求中的实体，因而PUT能够让文件复制到服务器，在GET请求中文件能够被复制到客户端。与之相反，POST中URI标识的程序处理请求中的实体，因此通常应用于交互式程序。PUT用法很多，如上传内容到网站，这种情况下必须加以认证。但是，在站点上存储文件通常使用其他方式，如FTP。

**TRACE**

客户端通过这种方法接收发至服务器的请求，用于诊断目的。

# 参考

TCP/IP Guide

部分内容来源于网络

# 网络基本功（二十三）：Wireshark抓包实例诊断TCP连接问题

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese




## 介绍

前文论述了TCP基础知识，从本节开始，通过TCP抓包实例来诊断TCP常见问题。

TCP进程通讯时，双方打开连接，发送数据，最后关闭连接。当TCP打开连接时，从源端口到目的端口发送一个请求。在应用建立或关闭时可能发生一些问题。本文讨论用Wireshark网络抓包的方法来定位及解决这一问题。

## 更多信息

问题的表现形式:

问题可能有多种表现类型：

- 尝试运行应用程序但发现应用程序无法工作。尝试浏览网络但无法获得响应。
- 尝试发送邮件但无法连接到邮件服务器。
- 问题可能由简单原因引起，如服务器宕机，服务器上没有运行应用程序，或在客户端到服务器的某一处网络断开。
- 问题也可能由复杂原因引起，如DNS问题，服务器内存不足无法连接（例如某一应用占用高内存空间），重复IP，以及其他原因。

处理方法:

下文会介绍解决问题的线索以及如何通过抓包来诊断TCP连接问题。通常，这些问题会导致运行应用程序时无法得到任何结果。

当你在运行一个应用程序时，例如数据库客户端，邮件客户端，观看视频等等，而又无法获得输出，按照以下步骤诊断：

1. 确认服务器和应用程序正在运行。
2. 确认客户端正在运行，IP地址已配置（手动或通过DHCP），并连接至网络。
3. Ping服务器并确认连接正常。
4. 在某些情况下，ping不通服务器但连接正常。这是由于防火墙拦截了ICMP信息，所以如

果无法ping通并不一定表示连接有问题。防火墙可能是网络中的专用设备或Windows/Linux/UNIX终端设备上安装的防火墙。

5. 抓包文件中，查找以下模式：

- 三重SYN信息而没有响应（见以下截屏）
- SYN信息带一个reset(RST)响应

这两种情况下都有可能是防火墙拦截了特定应用程序或应用程序没有在运行。

以下截屏是一个简单的case：客户端无法连接到web服务器81.218.31.171（报文61,62和63）。可能是由于不被防火墙允许，或服务器发生故障。可以看到另一个站点108.160.163.43（报文65,66和67）的连接正常，因此连接问题仅限于81.218.31.171。

```
Connection not opened to 81.218.31.171 (SYN / SYN / SYN)
61 5.088214000  10.0.0.3        81.218.31.171   TCP  62 51910 > http [SYN] Seq=0 Win=8192
62 5.090244000  10.0.0.3        81.218.31.171   TCP  62 51909 > http [SYN] Seq=0 Win=8192
63 5.178158000  10.0.0.3        81.218.31.171   TCP  62 51912 > http [SYN] Seq=0 Win=8192
64 6.247500000  10.0.0.3        173.194.78.125  TCP  66 51919 > xmpp-client [SYN] Seq=0 Wi
65 6.449442000  10.0.0.3        108.160.163.43  TCP  66 51921 > http [SYN] Seq=0 Win=8192
66 6.480809000  108.160.163.43  10.0.0.3        TCP  66 http > 51921 [SYN, ACK] Seq=0 Ack=
67 6.480936000  10.0.0.3        108.160.163.43  TCP  54 51921 > http [ACK] Seq=1 Ack=1 Win
68 6.481512000  10.0.0.3        108.160.163.43       343 GET /subscribe?host_int=3408558264
69 6.512241000  108.160.163.43  10.0.0.3             54 http > 51921 [ACK] Seq=1 Ack=290 W
70 6.512988000  108.160.163.43  10.0.0.3             443 HTTP/1.1 200 OK (text/html)
Connection opened to 108.160.163.43 SYN / SYN-ACK / ACK
```

下例是一个这种情况相对复杂的case。该case中，客户想要登录到camera服务器来访问远程站点的camera。camera服务器的IP地址为135.82.12.1，问题在于客户能够看到服务器主页上的登录窗口，但无法登进系统。在下面的截图中可以看到，打开了一个到IP地址135.82.12.1的连接。到HTTP服务器的TCP连接是打开的，一开始看上去没有连接问题：

```
Filter: ip.addr==135.82.12.1   Expression... Clear Apply Save
No.  Time       Source       Destination  Protocol Length Info
2113 17.665372  10.0.0.3     135.82.12.1  TCP   66 62423 > http [SYN] Seq=0 Win=8
2120 17.746627  135.82.12.1  10.0.0.3     TCP   66 http > 62423 [SYN, ACK] Seq=0
2121 17.746693  10.0.0.3     135.82.12.1  TCP   54 62423 > http [ACK] Seq=1 Ack=1
2122 17.747085  10.0.0.3     135.82.12.1  HT   316 GET / HTTP/1.1
2130 17.862143  135.82.12.1  10.0.0.3          54 http > 62423 [ACK] Seq=1 Ack=2
2189 18.736301  135.82.12.1  10.0.0.3         145 [TCP segment of a reassembled
2191 18.767301  135.82.12.1  10.0.0.3        1466 [TCP segment of a reassembled
Connection opened to IP Address 135.82.12.1 TCP Port 80 (http)
```

当我们过滤出目的IP地址为135.82.12.1的数据流，也就是camera服务器。这里可以看到，当尝试连接TCP端口6036时，得到了一个RST/ACK响应，有以下可能性：

- 防火墙拦截了端口6036
- 如果配置了端口地址转换（PAT），那么仅转换端口80而非6036
- 用户名和密码验证是在TCP端口6036上完成的，防火墙仅允许端口80，验证被拦截，应用无法工作

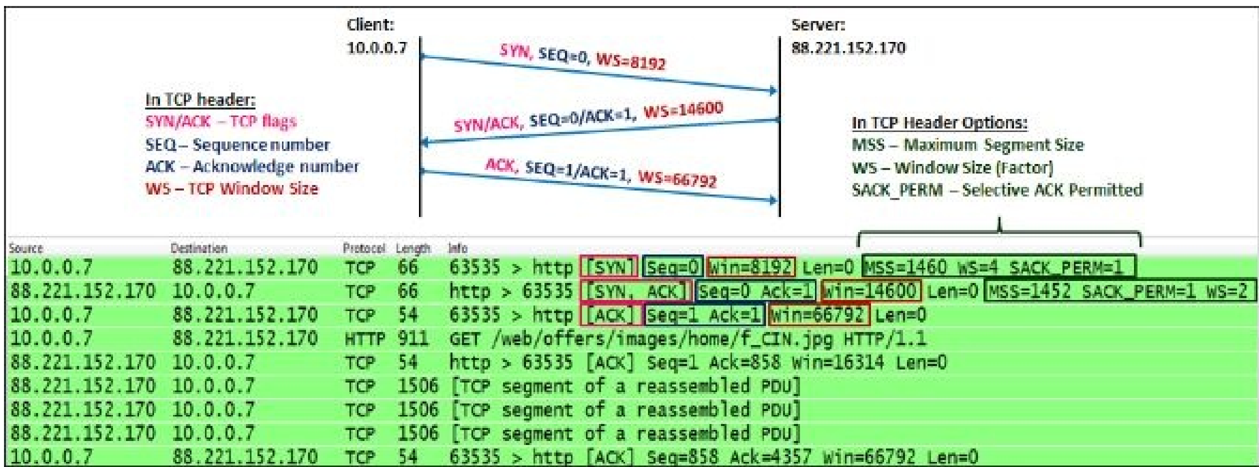
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 2620 | 36.423135 | 10.0.0.3 | 135.82.12.1 | TCP | 54 | 62438 > http [ACK] Seq=915 |
| 2 | | 10.0.0.3 | 135.82.12.1 | TCP | 66 | 62442 > 6036 [SYN] Seq=0 W |
| 2 | | 135.82.12.1 | 10.0.0.3 | TCP | 54 | 6036 > 62442 [RST, ACK] Se |
| 2 | | fe80::c067:2c23:335: | ff02::c | SSDP | 208 | M-SEARCH * HTTP/1.1 |
| 2 | | 10.0.0.3 | 194.90.1.5 | ICMP | 74 | Echo (ping) request id=0x |
| 2 | | 194.90.1.5 | 10.0.0.3 | ICMP | 74 | Echo (ping) reply id=0x |
| 2626 | 37.329129 | 10.0.0.3 | 135.82.12.1 | TCP | 62 | 62442 > 6036 [SYN] Seq=0 W |
| 2627 | 37.369547 | 135.82.12.1 | 10.0.0.3 | TCP | 54 | 6036 > 62442 [RST, ACK] Se |
| 2628 | 38.023274 | 10.0.0.3 | 194.90.1.5 | ICMP | 74 | Echo (ping) request id=0x |

Connection Trials to TCP port 6036:
- SYN request
- RST/ACK response

总之，当无法正常连接服务器时，检查服务器和客户端是否所有**TCP/UDP**端口都能通过网络转发，以及是否有未知的端口。

工作过程：

TCP连接开始时，发生了以下三步：




1. 客户端TCP进程发送了一个SYN报文。该报文中SYN标志位设置为1。这一报文中客户端：

- 指定自己的初始序列号。这是客户端发送给服务器的第一个字节。
- 指明自己的窗口大小。这是客户端分配给进程的缓存大小（位于客户端的RAM）。
- 设置自己将要使用的选项：MSS，Selective ACK，等等。

2. 当服务器收到建立连接请求，服务器：

- 发送SYN/ACK给客户端，确认接收到SYN请求。
- 指明服务器端的初始序列号。这是服务器发送给客户端的第一个字节。
- 指明服务器的窗口大小。这是服务器分配给进程的缓存大小（位于服务器RAM）。
- 回复请求选项并设置服务器端选项。

3. 当接收到服务器的SYN/ACK，客户端：

- 发送ACK报文给服务器，确认从服务器接收到SYN/ACK.
- 指明客户端窗口大小。尽管这一参数在第一个报文中定义过了，服务器还是会参考这个值，因为这是最新的窗口大小。

在TCP头部的选项字段中，有以下几个主要选项：

- Maximum Segment Size（MSS）：TCP数据报的最大字节数，即从TCP头部开始直到报文末尾的字节数。
- Windows Scale Option (WSopt)：这一因子与TCP头部的Window Size字段相乘，通知接收方扩大缓存。由于头部最大窗口大小是64KB，乘以因子4也就是256KB窗口大小。
- SACK：Selective ACK，该选项使连接双方能够仅确认指定报文，当单个报文丢失，只有这个报文会被重传。连接建立时，双方都需要同意SACK。
- Timestamps Option（TSopt）：该参数指客户端和服务器之间的延时。

在这一阶段，双方：

- 同意建立连接
- 知道对方的初始序列号
- 知道对方的窗口大小

  在建立连接时，除了三路握手信号之外，其他都表示有问题。包括SYN没有响应，SYN之后SYN/ACK最后没有ACK，SYN响应为RST，等等。

总结：

- 如果**SYN**报文收到回复**RST**，则检查拦截了**port**号的防火墙。
- 三次**SYN**而没有任何回复，或者是由于应用程序没有响应，或者是由于防火墙拦截了特定端口上的请求。
- 永远记住确认一下是否有**NAT**，端口转发，以及涉及**TCP**和**UDP**端口的机制。这些机制可能会中断**TCP**正常操作。

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十四）：Wireshark抓包实例分析TCP重传

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese




## 介绍

TCP发送一个或一组报文，会等待收到报文的确认信息。重传，即发生在报文没有到达或确认信息没有及时返回的情况下。当发现网速变慢时，原因之一可能就是重传。发生重传的原因有多种，在客户机或服务器两边端口应用Wireshark有助于诊断问题。本文通过抓包实例阐述各种可能性。

## 更多信息

诊断过程:

1. 在相应端口开始抓数据。
2. 找到**Analyze | Expert Info**菜单。
3. 在**Notes**之下，查找**Retransmission**。
4. 点击（+）符号即可打开重传列表。鼠标点击各行可在抓包面板看到重传报文。
5. 现在问题来了，怎样定位问题呢？
6. 通过以下方式查看重传来自哪里：
    - 在Expert Info窗口一个一个查看报文，在抓包面板查看哪些是重传报文（适合于有经验的用户）
    - 在报文面板，配置显示过滤器expert.message == “Retransmission (suspected)”，即可看到抓包文件中所有重传报文
    - 应用过滤器，在**Statistics & Conversations**窗口查看**Limit to display filter**部分。

Case 1：重传至多个目的地址

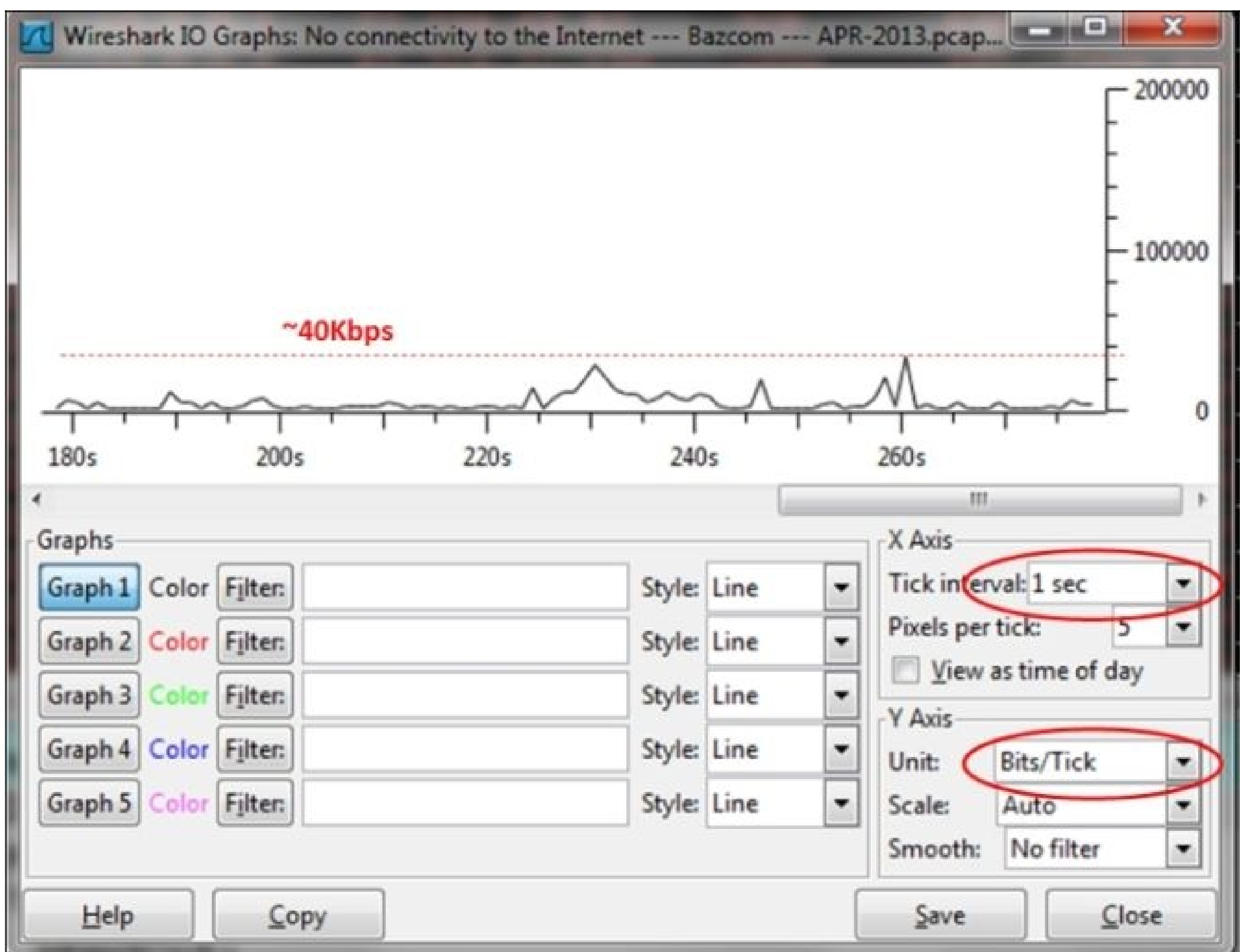
以下截屏中，可看到有多次重传，分布于多台服务器，目的端口号为80（HTTP）。也可以发现重传由端口10.0.0.5发送，因此报文是丢失在发往Internet的途中，或确认信息没有及时从web服务器发回。

Filter: expert.message == "Retransmission (suspected)" Various destination addresses Save Retransmissions All retransmissions to port 80 (HTTP)

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 96 | 29.203230000 | 10.0.0.5 | 212.199.184.51 | HTTP | 366 | [TCP Retransmission] GET /iefilter.html HTTP/ |
| 98 | 29.312500000 | 10.0.0.5 | 94.127.73.180 | TCP | 783 | [TCP Retransmission] 55317 > http [PSH, ACK] |
| 99 | 29.343596000 | 10.0.0.5 | 77.234.43.92 | TCP | 638 | [TCP Retransmission] 55310 > http [PSH, ACK] |
| 100 | 29.390393000 | 10.0.0.5 | 81.218.31.136 | HTTP | 491 | [TCP Retransmission] GET /24x24/30.png HTTP/1 |
| 101 | 29.390393000 | 10.0.0.5 | 108.168.157.82 | TCP | 1506 | [TCP Retransmission] 55326 > http [ACK] Seq=1 |
| 103 | 29.437199000 | 10.0.0.5 | 74.125.232.145 | TCP | 1070 | [TCP Retransmission] 55320 > http [PSH, ACK] |
| 106 | 29.530780000 | 10.0.0.5 | 74.125.232.145 | TCP | 1071 | [TCP Retransmission] 55321 > http [PSH, ACK] |
| 110 | 29.811579000 | 10.0.0.5 | 212.199.184.51 | HTTP | 366 | [TCP Retransmission] GET /iefilter.html HTTP/ |
| 114 | 30.513564000 | 10.0.0.5 | 94.127.73.180 | TCP | 590 | [TCP Retransmission] [TCP segment of a reasse |
| 115 | 30.544753000 | 10.0.0.5 | 77.234.43.92 | TCP | 590 | [TCP Retransmission] [TCP segment of a reasse |
| 116 | 30.591557000 | 10.0.0.5 | 81.218.31.136 | HTTP | [illegible] | [TCP Retransmission] GET /24x24/30.png HTTP/1 |
| 117 | 30.638349000 | 10.0.0.5 | 74.125.232.145 | TCP | | ransmission] [TCP segment of a reasse |
| 118 | 30.638366000 | 10.0.0.5 | 108.168.157.82 | TCP | | ransmission] 55326 > http [ACK] Seq=1 |
| 121 | 30.731954000 | 10.0.0.5 | 74.125.232.145 | TCP | | ransmission] [TCP segment of a reasse |
| 125 | 31.012738000 | 10.0.0.5 | 212.199.184.51 | HTTP | 366 | [TCP Retransmission] GET /iefilter.html HTTP/ |
| 158 | 31.387223000 | 10.0.0.5 | 81.218.31.136 | HTTP | 489 | [TCP Retransmission] GET /shadow.png HTTP/1.1 |
| 159 | 31.387356000 | 10.0.0.5 | 81.218.31.136 | HTTP | 503 | [TCP Retransmission] GET /save-center rollove |

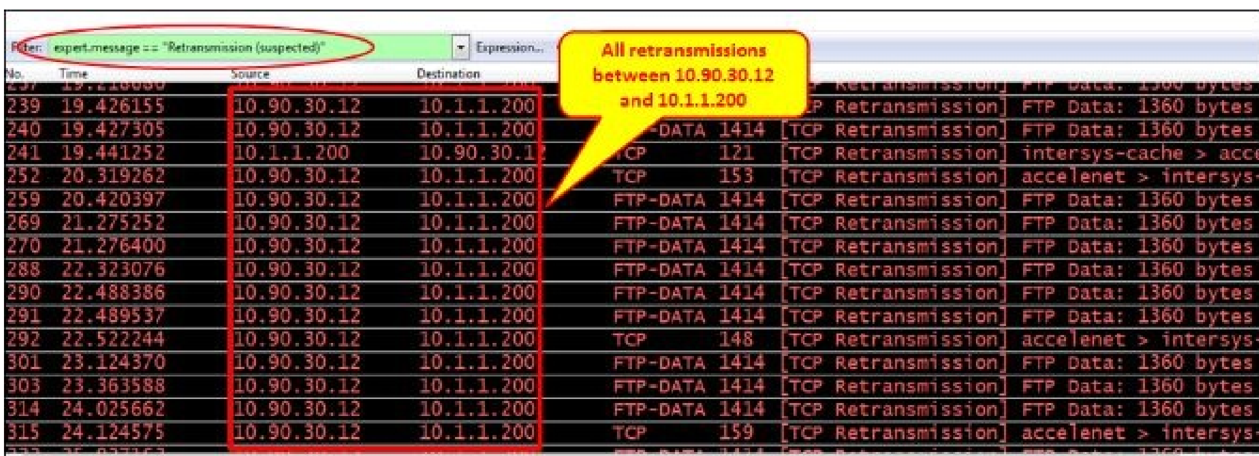
问题发生在发往Internet的线路上，怎样知道是什么问题呢？

1. 从**Statistics**菜单，打开**IO Graph**。
2. 本例中，可看到链路负载非常低。可能是有故障，或有另一条高负载链路。
3. 可以通过登录到通信设备或SNMP浏览器查看引起重传的原因：报文丢失及错误。参考以下截屏：



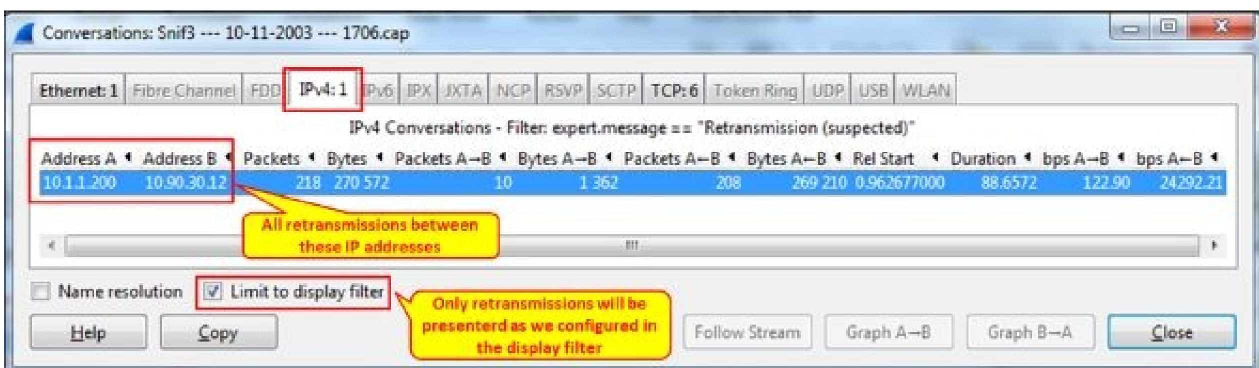

Case 2：重传至单一连接

如果所有重传发生于同一IP，同一TCP端口号，则可能是慢速应用引起。看以下截屏：

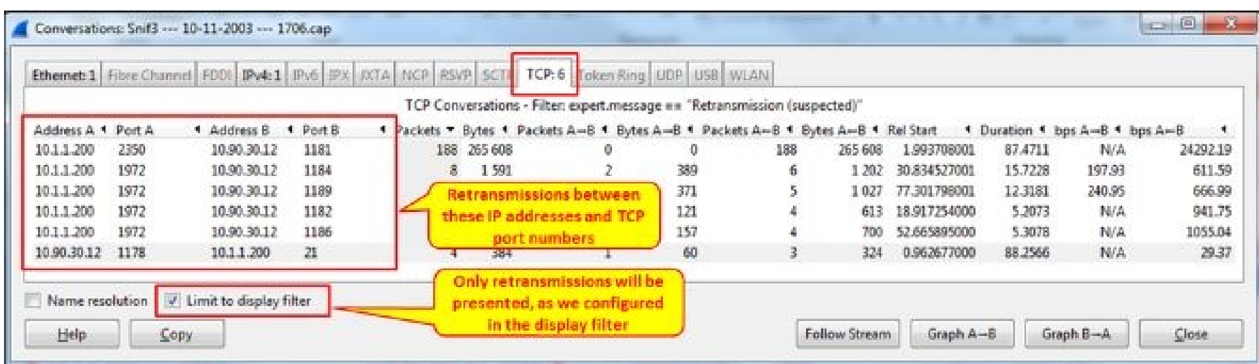



对于单一连接的重传，进行以下操作：

1. 从Statistics菜单打开Conversations，选择Limit to display filter，可以看到所有发生重传的会话，本情况下，是一个单一会话。
2. 如下图所示，通过选择**IPv4**标签可看到从哪个IP地址重传：




3. 如下图所示，通过选择TCP标签看到重传来自哪一端口：



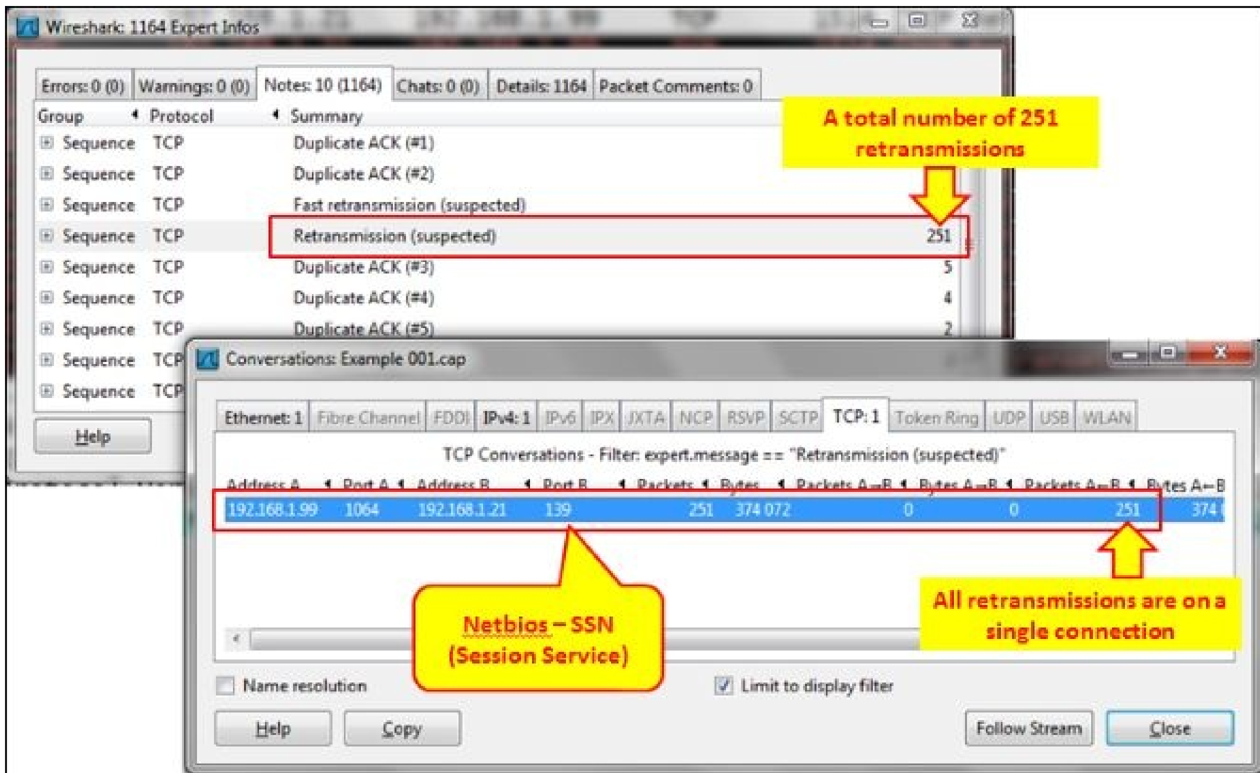

要定位问题，进行以下步骤：

1. 查看IO graph，确保链路不忙。（链路忙的表征例如流量接近带宽。例如，带宽为10Mbps，在IO graph中看见流量接近10Mbps，这就表明链路负载较高。不忙的链路IO会有很多高低起落，峰值以及空闲间隙）。
2. 如果链路不忙，则可能是服务器对于IP地址10.1.1.200有问题（10.90.30.12发送了绝大多数重传，所以可能是10.1.1.200响应较慢）
3. 从报文面板可以看出应用是FTP数据。有可能FTP服务器工作于active模式。因此在端口

2350打开连接，服务器将端口更改为1972，所以可能是慢速FTP软件响应问题引起的重传。

Case 3：重传模式

观察TCP重传的一个重要考量是是否能看出一些重传模式。在以下截屏中，可以看见所有重传来自单一连接，位于客户端与服务器的NetBIOS会话服务（TCP端口139）。




看起来像一个简单的服务器/客户端问题，但查看抓包面板，如下图所示：

Filter: expert.message == "Retransmission (suspected)" Expression... Clear Apply Save

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1477 | 0.040695 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1505 | 0.032809 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1536 | 0.036683 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1563 | 0.030141 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1594 | 0.036375 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1627 | 0.039216 | .168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1655 | 0.033171 | 192. 1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1686 | 0.036725 | 192 | 92.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1713 | 0.029351 | 192 | 92.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1744 | 0.036509 | 192 | 92.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1768 | 0.421340 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |
| 1790 | 0.023597 | 192.168.1.21 | 192.168.1.99 | TCP | 1514 | [TCP Retransmission] |

Retransmissions with 30-35mS between in between

可以看见重传总是周期性的每30ms发生一次。问题是由于客户端在软件中执行了财务进程，导致软件每30-36ms就减速一次。

Case 4：应用无响应导致重传

另一个可能导致重传的原因是客户端或服务器没有响应请求。这种情况下，会看到五次重传，时间也会逐渐延长。五次连续重传后，发送方认为连接断开（某些情况下，会发送reset来关闭连接，取决于软件实施）。断开连接之后，可能会发生两件事情：

- 发送SYN请求至客户端，以打开一个新的连接。这种情况下用户会看到应用冻结，过了15-20秒之后重新开始工作。
- 不发送SYN，用户需要重新运行应用程序（或应用程序的一部分）

下图显示了打开新连接的情况：

Time intervals increase with every retransmission

Five consecutive retransmissions

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1159 | 0.000406 | 192.16 | 192.168.201.93 | TCP | 60 | http > tscchat [ACK] Seq=220556 Ack=2920 |
| 1160 | 0.220322 | 192.168.201.93 | 192.168.3.50 | TCP | 590 | [TCP Retransmission] [TCP segment of a r |
| 1161 | 0.656270 | 192.168.201.93 | 192.168.3.50 | TCP | 590 | [TCP Retransmission] [TCP segment of a r |
| 1162 | 1.203085 | 192.168.201.93 | 192.168.3.50 | TCP | 590 | [TCP Retransmission] [TCP segment of a r |
| 1163 | 2.406248 | 192.168.201.93 | 192.168.3.50 | TCP | 590 | [TCP Retransmission] [TCP segment of a r |
| 1164 | 4.812443 | 192.168.201.93 | 192.168.3.50 | TCP | 590 | [TCP Retransmission] [TCP segment of a r |
| 1165 | 9.625596 | 192.168.201.93 | 192.168.3.50 | TCP | 62 | agentview > http [SYN] Seq=0 Win=65535 L |
| 1166 | 0.004414 | 192.168.3.50 | 192.168.201.93 | TCP | 60 | http > agentview [SYN, ACK] Seq=0 Ack=1 |
| 1167 | 0.000033 | 192.168.201.93 | 192.168.3.50 | TCP | 54 | agentview > http [ACK] Seq=1 Ack=1 Win=6 |
| 1168 | 0.000164 | 192.168.201.93 | 192.168.3. | TCP | 590 | [TCP segment of a reassembled PDU] |

A new connection established

Case 5：由于延时变化导致重传

TCP能够充分容忍延时，前提是延时大小不发生变化。当延时改变时，就会发生重传。诊断是否由该原因引起的方法：

1. 第一件事是ping目的地址，并且得到检查通信链路延时的第一条信息。
2. 检查延时变量，可能由以下原因引起：
    - 由于不稳定或繁忙通信链路引起。这种情况下，可以看到ping命令的延时变化，通常由于带宽较窄。
    - 由于应用过载或资源不足，这种情况下，只有该应用发生很多重传。
    - 通信设备过载（CPU,缓存）引起延时。检查方式直接连接通信设备。
3. 使用Wireshark工具诊断延时问题。

如果重传达到0.5个百分比，性能就会下降，断开连接将会达到5个百分比。这取决于应用及其对于重传的敏感性。

定位重传问题

当你看到通信链路上发生重传，进行以下步骤：

1. 定位问题——是一个特定IP地址，特定连接，特定应用，还是其他问题。
2. 查看问题是否由于通信链路，丢包，慢速服务器还是PC。查看应用是否慢速。
3. 如果不是由于上述原因，检查延时变化。

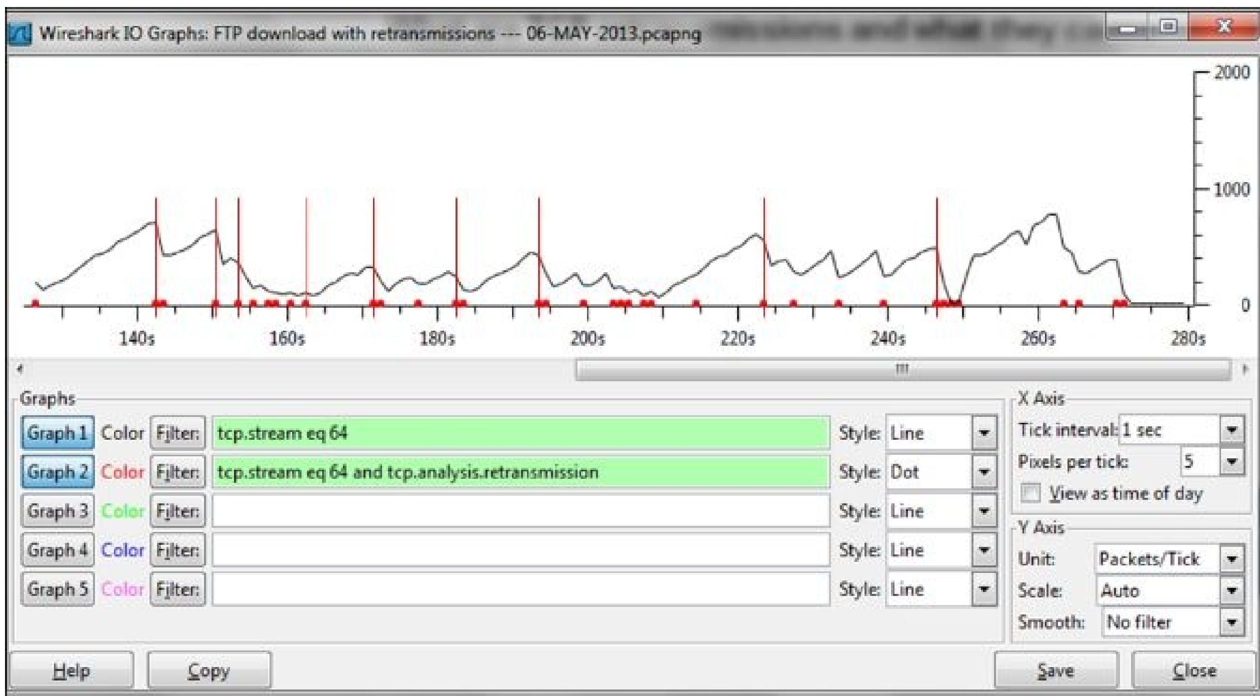
工作原理:

TCP序列号/确认机制详见前文：网络基本功（十）：细说TCP确认机制 。那么重传是由什么原因引起呢？当报文确认信息丢失，或ACK没有及时到达，发送方会进行以下两步操作：

1. 再次发送报文
2. 减少吞吐量。

更多TCP重传内容详见前文：网络基本功（九）：细说TCP重传 。

以下图中展示了重传减少发送方吞吐量（红色细线）：




# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十五）：Wireshark抓包实例分析TCP重复ACK与乱序

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese


## 介绍

TCP的一大常见问题在于重复ACK与快速重传。这一现象的发生也是由于性能问题，本章讨论如何发现这一问题以及他们意味着什么。

另一个常见问题是前一片段丢失以及乱序片段。某些情况下，这一现象喻示着故障发生，可能是由于网络问题或是抓包中断。

## 更多信息

重复**ACK**与快速重传**:**

当网速变慢时，重复ACK是可能的原因之一。大多数情况下，重复ACK的发生是由于高延时，延迟的变化，或无法响应ACK请求的慢速终端。

1. 当重复ACK的数量保持在合理范围时，即1或2个百分比，则可能不是本机问题。
2. 当有大量的重复ACK时（假设有10个），则可能：
    - 通信链路繁忙引起延迟改变
    - 服务器或客户端无响应
3. 快速重传是对重复ACK的响应报文。
4. 下图是该问题的示例。本例中51个重复ACK之后发生了快速重传：

5. 以下是如何解决该问题：

   - 如果重复ACK和重传数量较少（少于1个百分比），是可以接受的。
   - 如果重复ACK发生在无线网络环境，或是Internet之上的连接，延时或是延时的改变对于这类网络来说很常见，所以也没有什么可做的。
   - 如果发生在组织内的网络，则可能有问题。如果发生在LAN之上，检查严重的问题，例如缓存和CPU负载，慢速服务器，等等。如果发生在WAN之上，查看延时，负载以及线路不稳定。

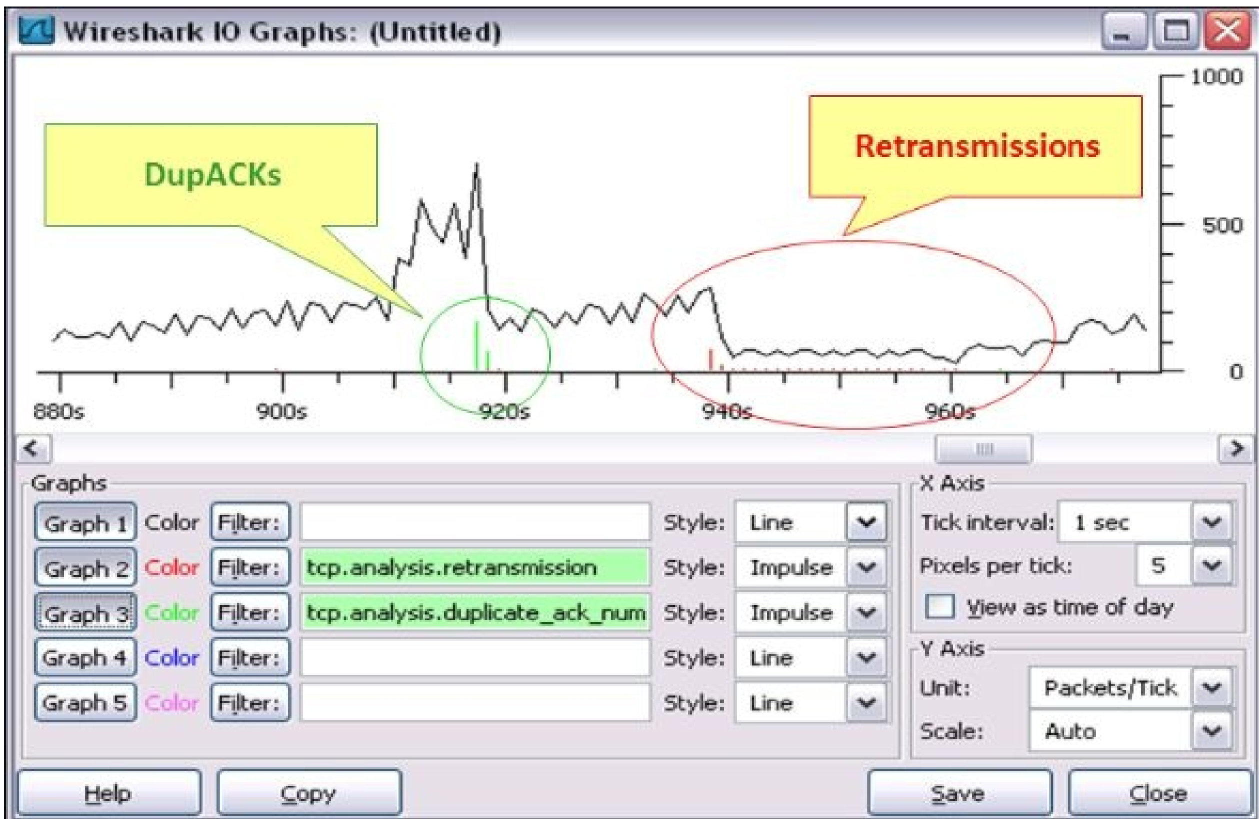
工作原理

当发现有丢失报文时（期望的序列号没有收到），或者收到了预期之外的序列号。这种情况下，接收端生成一个ACK，声明自己希望收到的下一个序列号。接收方持续生成丢失片段的ACK请求，直到实际收到。

在发送方，当它收到三个相同的ACK（初始ACK和两个重复ACK），就会假设有报文丢失并重传该报文，无论重传计时器是否过期。再次发送的报文称为快速重传。

重复ACK也减少了发往网络的吞吐量。减少了多少吞吐量取决于TCP版本。比较早期的TCP版本中出现了重复ACK，发送方将吞吐量减少为之前的一半。在多个DupACK的情况下，吞吐量减到最小。

下图显示了重复ACK和重传的典型例子，本图中第一次重复ACK将吞吐量降低至大约40%，之后重传将吞吐量减至最小。




乱序报文:

在两端抓包，乱序情况下需要关注三种现象：

- 先前片段丢失：当前收到报文的序列号高于该连接的下一个期望序列号时，表明之前的一个或多个报文未能到达
- 乱序报文：当前报文的序列号低于该连接先前收到的报文
- 先前片段未能捕捉：（Wireshark 1.8.x及以上版本）：同先前报文丢失。

何时发生？

用户可能在以下情况看到乱序报文：

- 连接开始时抓包：当建立连接时抓包，这时，看到连接上的报文没有SYN/SYN-ACK/ACK，因此，Wireshark认为连接有问题。

- 确实有报文丢失：这时会看到丢失报文重传和/或重复ACK告知发送方重传丢失报文。

| No. | Source | Destination | Protocol | Length | Info | SEQ |
|---|---|---|---|---|---|---|
| 330 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP segment of a reassembled PDU] | 312401 |
| 331 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#54] 57999 > http | 369 |
| 332 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP Previous segment not captured] | 319501 |
| 333 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#55] 57999 > http | 369 |
| 334 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP segment of a reassembled PDU] | 320921 |
| 335 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#56] 57999 > http | 369 |
| 336 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP Previous segment not captured] | 326601 |
| 337 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#57] 57999 > http | 369 |
| 338 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP segment of a reassembled PDU] | 328021 |
| 339 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#58] 57999 > http | 369 |
| 340 | 62.90.90.210 | 10.0.0.6 | TCP | 1474 | [TCP Previous segment not captured] | 332281 |
| 341 | 10.0.0.6 | 62.90.90.210 | TCP | 90 | [TCP Dup ACK 223#59] 57999 > http | 369 |

1420 Bytes
5680 Bytes (4*1420Bytes)
1420 Bytes
4260 Bytes (3*1420Bytes)

上图是报文丢失的典型示例。从图中可见，10.0.0.6尝试浏览站点62.90.90.210。这一过程中， TCP片段每个1420字节发送到web服务器，334到336之间3个报文丢失，338到340之间2个报文丢失。两者Wireshark都有提示：TCP’s previous segment is not captured.

- 延时变化：这可能是由于报文从源地址到目的地址经由不同的路由。检查这一点可以使用Tracert，在源和目的地址之间查找路由改变。如果在公司内部网络上是可以做到的，例如，在路由器上配置trap。
- 数据捕捉问题：可能报文正常收发，但Wireshark没有捕捉到。可能有以下几种原因：
  - 数据量比较大时，Wireshark在高比特率的情况下可能会丢失报文（高于150-180 Mbps）。要避免这一问题，使用其他工具（大多数需要付费）。
  - 台式机不够强大，内存或CPU无法让Wireshark工作的足够快。这一点很好发现。
  - 当LAN交换机的端口缓存太小，报文可能被丢弃。连接到交换机（用控制台或telnet连接）使用交换机命令行来检查该问题。
  - 无线网络抓包，由于某种原因没有看到所有发送报文。

总结

乱序报文的原理很简单。TCP发送以其字节数为编号的报文到接收方。当一个报文没有按照顺序到达时，Wireshark就会注意到。原因有两点：

- 确实有问题：这时会看到重传和重复ACK，这是TCP对于收到乱序报文的响应。
- 抓包问题：这时仅看到乱序报文，但没有看到对可能丢失及乱序报文的响应，可能实际上并没有问题。

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十六）：Wireshark抓包实例分析TCP窗口及reset

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese


## 介绍

TCP最重要的机制之一是滑动窗口机制，以及用以控制TCP终端节点愿意接收的数据总量的流控机制。

TCP reset可以在几种情况下被发送。有一些是协议的正常工作过程，有一些则表明可能有问题。本节中，我们查找问题以及分析解决问题的方法。

本章讨论以上两个问题。

## 更多信息

**TCP窗口问题:**

TCP零窗口，零窗口探测，零窗口违例

TCP零窗口发生于接收方在TCP头部的window字段广播接收窗口零字节的时候。这一事件告知发送方停止发送数据，因为接收方缓存已满。这也表明接收方可能发生以下问题：

- 服务器无法为进程分配组后的内存
- 应用碰到没有足够内存的问题，因此TCP需告知发送方停止发送数据
- 应用消耗太多内存因此操作系统要限制应用资源

TCP零窗口探测由发送方发出，以查看接收方的零窗口是否依然存在。这一消息通过发送下一字节数据给接收方，如果接收方回复窗口大小仍然为零，则发送方的探测计时器加倍。

TCP窗口违例：发送方忽略接收方的零窗口大小并发送额外字节数据。TCP零窗口违例表明协议栈中有TCP错误。为了检查是何问题，检查是否有以下事件：

- 某一终端设备（服务器或客户机）报出终端设备故障
- 某一应用报出常规应用错误
- 应用中执行某一操作时报错，例如，打开一个表格，发送一份文件至打印机，创建一个报告，或其他操作。这种情况下，是应用问题。

TCP窗口更新

TCP将窗口更新发送至连接的对端，以表明缓存大小更改，并且准备好接受更高或更低的数据速率（缓存大小决定了发送方被允许的发送速率）。这一情况发生于：

- TCP接收方从零窗口中恢复，告知发送方重新发送速率。这一情况下，无需进行处理，只需检查第一次导致零窗口的问题。
- TCP接收方频繁更改窗口大小。该情况下检查接收方被干扰的原因。可能是应用问题，内存问题，或终端设备上的其他问题。

看到这类现象无需担心，这就是TCP的工作机制。

TCP窗口已满

这一信息表明已发送的报文会完全填满接收方的接收缓存。发生于接收方没有对先前接收到的数据发出任何ACK确认信息，因此，这将会成为发送方从接收方收到ACK之前的最后一个报文数据。

这一事件的触发原因与触发零窗口的原因相同，是服务器或应用无响应的标志。典型实例如下图所示：

| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 182995 | 192.168.2.138 | 192.168.1.58 | TCP | 590 | [TCP segment of a reassembled PDU] |
| 182996 | 192.168.2.138 | 192.168.1.58 | TCP | 590 | [TCP segment of a reassembled PDU] |
| 182997 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | http > 47185 [ACK] Seq=1 Ack=131685 Win=1 |
| 183816 | 192.168.2.138 | 192.168.1.58 | TCP | 197 | [TCP Window Full] [TCP segment of a reass |
| 183828 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindow] http > 47185 [ACK] Seq=1 |
| 183966 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | [TCP ZeroWindowProbe] [TCP segment of a r |
| 183967 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindowProbeAck] [TCP ZeroWindow] |
| 184128 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | [TCP ZeroWindowProbe] [TCP segment of a r |
| 184129 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindowProbeAck] [TCP ZeroWindow] |
| 184388 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | [TCP ZeroWindowProbe] [TCP segment of a r |
| 184389 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindowProbeAck] [TCP ZeroWindow] |
| 185002 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | [TCP ZeroWindowProbe] [TCP segment of a r |
| 185003 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindowProbeAck] [TCP ZeroWindow] |
| 186460 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | [TCP ZeroWindowProbe] [TCP segment of a r |
| 186461 | 192.168.1.58 | 192.168.2.138 | TCP | 54 | [TCP ZeroWindowProbeAck] [TCP ZeroWindow] |
| 189877 | 192.168.2.138 | 192.168.1.58 | TCP | 60 | 47185 > http [RST, ACK] Seq=131828 Ack=1 |

上图中可以看到：

1. 报文183816，192.168.2.138告知192.168.1.58发送窗口已满。
2. 下一个报文，192.168.1.58发送一个信号至192.168.2.138，告知对方停止发送数据。这是一个零窗口信号。
3. 双方继续发送零窗口以及零窗口探测。
4. 连接的最后一个报文是192.168.2.138发送的RST报文，目的是断开连接。
5. 某些情况下零窗口可以通过窗口更改信息来回复。某些情况下可通过reset来关闭（可以是应用零窗口从而没有收到任何数据导致）。

工作原理

TCP滑动窗口机制如下：

1. 连接建立之后，发送方将数据发送至接收方，填入接收窗口。
2. 若干报文之后，接收方发送ACK至发送方，确认接收到其发送的字节数。发送ACK将接

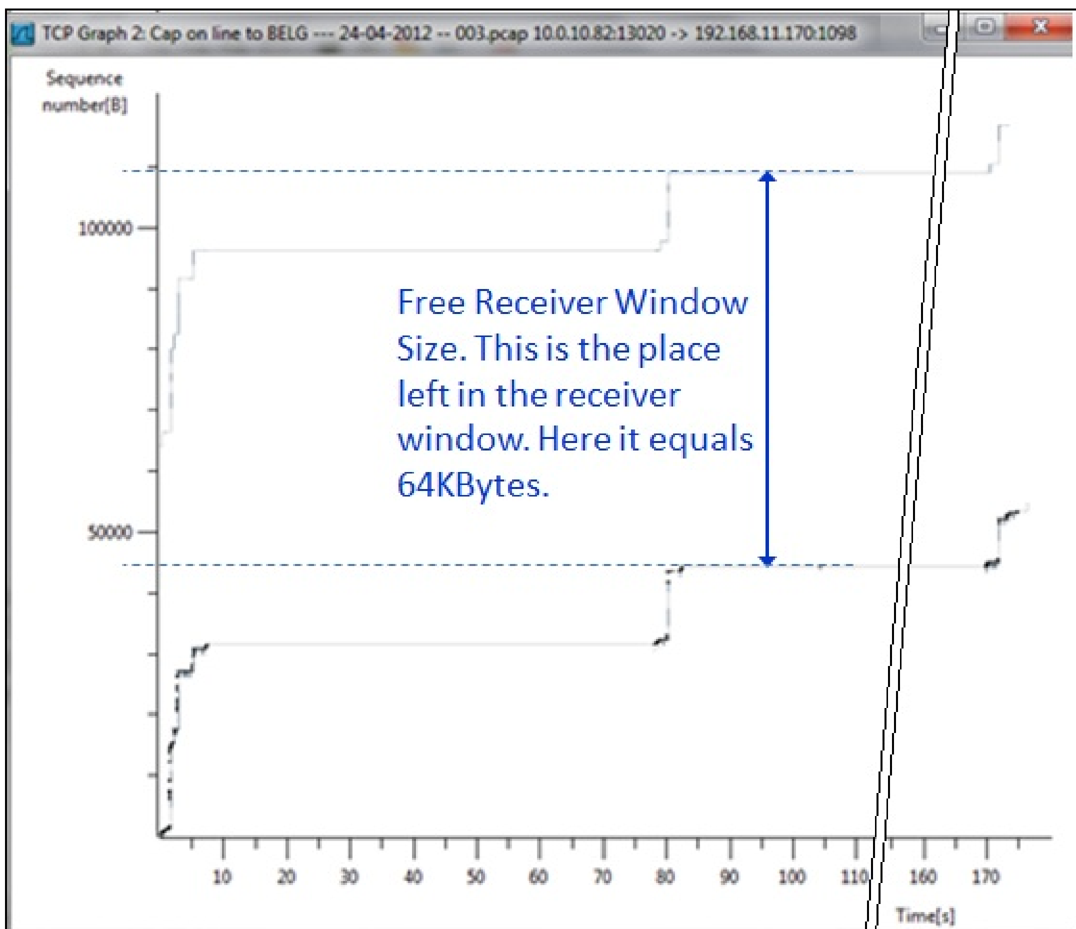
收窗口清零。

3. 这一过程持续下去，发送方向窗口中填入数据，接收方清空并发送确认信息。
4. 扩大接收窗口大小告知发送方增加吞吐量，减小窗口告知对方减小吞吐量。这一机制按照WS/RTT规则（随着TCP版本不同而有所改变）：

$$\text{Throughput [Bytes/Sec]} = \frac{\text{Window Size [Bytes]}}{\text{RTT [Sec]}}$$

- ✔ Throughput - the effective Bytes/Sec send by an application on a TCP connection
- ✔ Window Size - the TCP receiver window size
- ✔ RTT - the Round Trip Time between the sender and the receiver

也可以通过TCP吞吐量图表和IO图来查看问题。在TCP吞吐量图表中，使用TCP trace图表，上面一行显示了窗口大小，与下面一行的距离表明窗口的剩余大小。没有距离表示零窗口。




两行之间的固定距离表明接收方工作良好。当两行渐渐靠近，表明发送方速度高于接收方。只要这两行没有重叠，TCP就会继续发送数据。

**TCP reset及原因:**

在可疑的链路或服务器两端连接Wireshark，开始抓包。观察抓包窗口的每一个窗口信息。TCP reset可以在几种情况下被发送。有一些是协议的正常工作过程，有一些则表明可能有问题。本节中，我们查找问题以及分析解决问题的方法。

reset是用以告知接收方断开连接的TCP信号，通过将RST标志位置1来发送。正常的操作过程中，TCP通过SYN信号打开连接，通过FIN信号关闭连接。TCP的一大特性在于有问题时，或只是为了更好的性能，它能够快速关闭连接。

无故障时发送reset

TCP关闭连接的标准方式是通过FIN和FIN-ACK信号。为了关闭连接，用户需要四个报文：来自一方的FIN/ACK和ACK，以及另一方的同样报文。当你打开一个网页，可能同时打开了数十个连接（主页，新闻，广告，定期更新的图片等），要关闭所有这些有时需要数百个FIN和FIN-ACK报文。为了防止其发生，web服务器在很多情况下会在发送请求数据之后用reset断开连接。这是标准的做法，并取决于应用程序。

有故障时发送reset

某些情况下reset表明有故障发生（并不一定是通信故障）：

- 防火墙发送的**reset**：当远端服务器尝试打开连接但没有结果时，也许会看到返回RST信号。这是防火墙阻隔连接的情况。下图中，可看到发送的每一个SYN都返回以RST。

| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 54 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | ftp-data > 39542 [RST, ACK] Seq=1 Ack= |
| 55 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 39020 > ftp [SYN] Seq=0 Win=5840 Len=0 |
| 56 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | ftp > 39020 [RST, ACK] Seq=1 Ack=1 Win |
| 57 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 56045 > ssh [SYN] Seq=0 Win=5840 Len=0 |
| 58 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | ssh > 56045 [RST, ACK] Seq=1 Ack=1 Win |
| 59 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 47648 > telnet [SYN] Seq=0 Win=5840 Le |
| 60 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | telnet > 47648 [RST, ACK] Seq=1 Ack=1 |
| 61 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 44370 > 24 [SYN] Seq=0 Win=5840 Len=0 |
| 62 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | 24 > 44370 [RST, ACK] Seq=1 Ack=1 Win= |
| 63 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 48264 > smtp [SYN] Seq=0 Win=5840 Len= |
| 64 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | smtp > 48264 [RST, ACK] Seq=1 Ack=1 Wi |
| 65 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 49404 > 26 [SYN] Seq=0 Win=5840 Len=0 |
| 66 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | 26 > 49404 [RST, ACK] Seq=1 Ack=1 Win= |
| 67 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 46880 > nsw-fe [SYN] Seq=0 Win=5840 Le |
| 68 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | nsw-fe > 46880 [RST, ACK] Seq=1 Ack=1 |
| 69 | 192.168.1.141 | 192.168.1.123 | TCP | 74 | 41799 > 28 [SYN] Seq=0 Win=5840 Len=0 |
| 70 | 192.168.1.123 | 192.168.1.141 | TCP | 54 | 28 > 41799 [RST, ACK] Seq=1 Ack=1 Win= |

- 由于收发一方有问题发送的**reset**：可能的原因如：
  - 五个连续没有收到ACK回复的重传。当发送方没有收到任何重传回复，它就会发送一个reset信号到对端，告知其断开连接。
  - 另一个原因是连接之上几分钟都没有任何数据（分钟数取决于系统默认）。打开连接的一方通常会发送reset（但并不总是会这样做，取决于实现方式）。

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十七）：Wireshark抓包实例分析HTTP问题(上)

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese


## 介绍

HTTP的问题可能是由于慢速服务器或客户端，TCP性能问题，本文讨论上述问题以及其他可能因素。

## 更多信息

诊断过程:

浏览网页性能变差的原因有很多，需要逐步分析。步骤如下：

1. 首先，不仅要确认网络负载状况，还要注意通信链路上的出错率，以及导致性能变差的最明显的表现；
2. 诊断TCP问题，检查以下细节：
    - 在Expert info窗口，确保没有太多重传以及重复ACK（百分之0.5至0.8尚可忍受）。
    - 确保HTTP连接上没有reset，可能由于防火墙或站点限制引发。
3. 确保没有以下DNS问题：
    - 慢速响应时间
    - 域名未找到

如果以上均不适用，就需要对HTTP深入研究。

注意：将网络和IT环境看作一个整体。对于慢速网络浏览应用，TCP问题亦不能分离于HTTP，DNS问题。可能是由于慢速HTTP服务器，因服务器的慢速响应而产生了TCP重传。或者，由于DNS慢速服务器，打开网页可能需要好几秒钟。一步步定位问题就好了。

当你第一次打开一个网页，可能需要几秒钟。在这种情况下，应当查看以下情况：

1. 检查线路是否过载
2. 检查线路延时（通过ping工具）
3. 查看错误代码，通常能看到浏览器报错原因，但并不总是能看到。
4. 配置过滤器http.response >= 400并查看有多少错误。以下章节，你会看到需要注意的示例。

**Informational codes：**

| Code | Status | Explanation | What to do |
| --- | --- | --- | --- |
| 100 | Continue | Request completed successfully and the session can continue. | - |
| 101 | Switching protocols | The server is changing to a different HTTP version. It will be followed by an Upgrade header. | - |

**Success codes：**

| Code | Status | Explanation | What to do |
| --- | --- | --- | --- |
| 200 | OK | Standard OK response. | - |
| 201 | Created | The request has been fulfilled and a new resource has been created. | - |
| 202 | Accepted | The request was accepted and is still in process. | - |
| 203 | Non-authoritative information | The request was received with content from another server, and it was understood. | - |
| 204 | No content | The request was received and understood, and the answer that is sent back has no content. | - |
| 205 | Reset content | This is a server request to the client to reset the data that was sent to it. | - |
| 206 | Partial content | Response for a partial document request. | - |

**Redirect codes：**

| Code | Status | Explanation | What to do |
|---|---|---|---|
| 300 | Multiple choices | The requested address refers to more than one file. It can happen, for example, when the resource has been removed, and the response provides a list of potential locations for it. | - |
| 301 | Moved permanently | The requested resource has been moved permanently. Future requests should be forwarded to the attached URI. | - |
| 302 | Moved temporarily (found) | Page has been moved temporarily, and the new URL is available. Usually, you will be automatically forwarded. | Usually, you will see a Found code, and then another GET to the URL indicated |
| 303 | See other | The response to the request can be found in a different URI. It should be retrieved using an HTTP GET to that resource. | - |
| 304 | Not modified | When a request header includes an if modified since parameter, this code will be returned if the file has not changed since that date. | - |
| 305 | Use proxy | The requested resource must be accessed through | Check what proxy is |

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十八）：Wireshark抓包实例分析HTTP问题(下)

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese


## 介绍

本文承接上文。

## 更多信息

**Client errors:**

| Code | Status | Explanation | What to do |
|---|---|---|---|
| 400 | Bad request | The request could not be understood by the server due to a syntax problem. The request should be modified by the client before resending to it. | Check the website address. This can also happen due to a site error. |
| 401 | Authorization required | The client is denied access due to the lack of authentication codes. | Check your username and password. |
| 402 | Payment required | Reserved for future use. | |
| 403 | Forbidden | The client is not allowed to see a specific file. This can be due to the server access limit. | Check the credentials. Also, there are fewer chances that the server is loaded. |
| 404 | Not found | The requested resource could not be found. | This can be because the resource was deleted, or it never existed before. It can also be due to URL misspellings. |
| 405 | Method not allowed | The method you are using to access the file is not supported or not allowed by the resource. | |
| | | Content generated by the | |

| | | | |
|---|---|---|---|
| 406 | acceptable | according to the client request. | browser. |
| 407 | Proxy authentication required | Request authentication is required before it can be performed. | The client must first authenticate itself with the proxy. |
| 408 | Request timed out | It took the server longer than the allowed time to process the request. | Check response time and load on the network. |
| 409 | Conflict | The request submitted by the client cannot be completed because it conflicts with some established rules. | Can be because you try to upload a file that is older that the existing one or similar problems. Check what the client is trying to do. |
| 410 | Gone | The URL requested by the client is no longer available from that system. | Usually this is a server problem. It can be due to a file that was deleted or location was forwarded to a new location. |
| 411 | Content length required | The request is missing itsContent-Length header. | Compatibility issue on a website. Change/update your browser. |
| 412 | Precondition failed | The client has not set up a configuration that is required for the file to be delivered. | Compatibility issue on a website. Change/update your browser. |
| 413 | Request entity too long | The requested file was too big to process. | Server limitation. |
| 414 | Request URI too long | The address you entered was overly long for the server. | Server limitation. |
| 415 | Unsupported media type | The file type of the request is not supported. | Server limitation. |

以下示例是一个简单的客户端报错。按照以下步骤进行操作：

1. 右键有报错的报文。
2. 选择**Follow TCP stream**，会看到以下窗口：

```
}
GET /poker-client/broadcast.htm HTTP/1.1  (1)
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/
x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://www.888poker.com/poker-client/promotions.htm  (2)
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB7.1; Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1) ; .NET CLR 1.1.4322; .NET CLR 2.0.50727;
OfficeLiveConnector.1.3; OfficeLivePatch.0.0; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729;
InfoPath.1)
Host: www.888poker.com  (3)

HTTP/1.1 404 Not Found  (4)
Date: Sun, 16 Oct 2011 09:11:58 GMT
Server: Microsoft-IIS/6.0
srv: 2344432
```

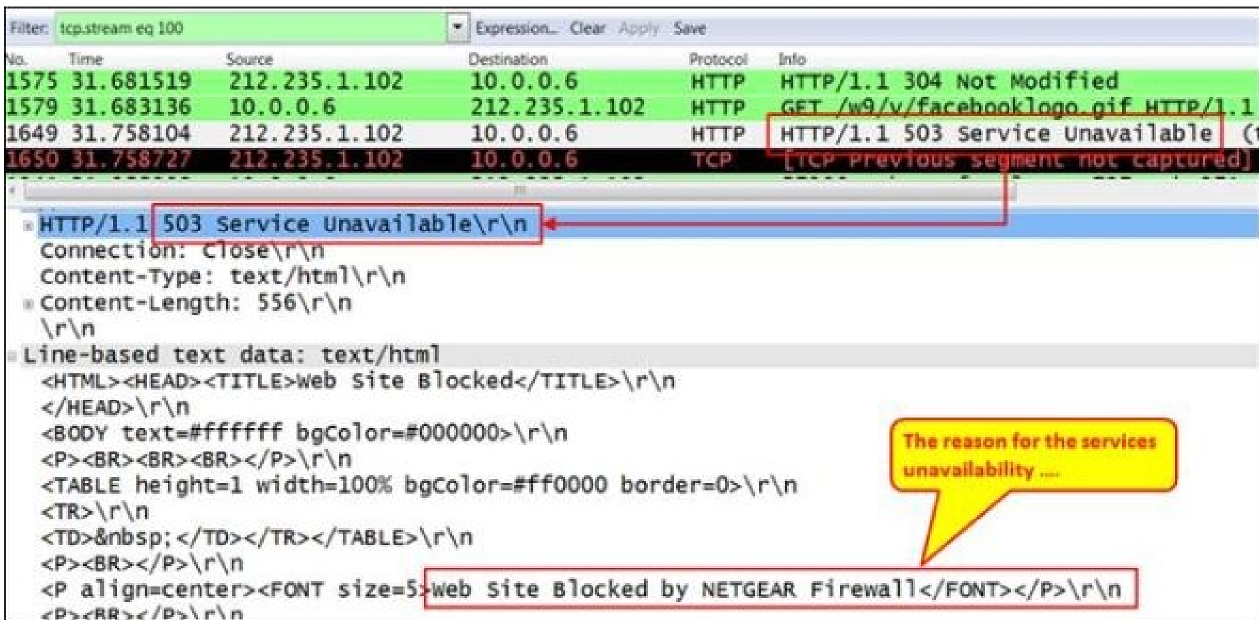
3. 显示以下内容：

- 客户端尝试浏览URI/poker-client/broadcast.htm（如截屏中1和3所示）
- URI通过http://www.888poker.com/poker-client/promotions.htm转发（截屏中2所示）
- 状态码为404 Not Found（如截屏中4所示）

**Client errors:**

| Code | Status | Explanation | What to do |
|---|---|---|---|
| 500 | Internal server error | The web server encountered an unexpected condition that prevented it from carrying out the client request for access to the requested URL. | Response that is usually caused by a problem in your Perl code when a CGI program is run. |
| 501 | Not implemented | The request cannot be executed by the server. | A server problem. |
| 502 | Bad gateway | The server you're trying to reach is sending back errors. | A server problem. |
| 503 | Service unavailable | The service or file that is being requested is not currently available. | A server problem. |
| 504 | Gateway timeout | The gateway has timed out. This message is like the 408 timeout error, but this one occurs at the gateway of the server. | Server is down or nonresponsive. |
| 505 | HTTP version not supported | The HTTP protocol version that you want to use for communicating with the server is not supported by it. | Server does not support the HTTP version. |

服务器不可用（错误代码503）可能有多种原因。以下示例是一个小办公室碰到的问题：员工能够访问Facebook，但当他们点击站点上的链接，则显示页面被拦截。以下截屏中，可看出页面被防火墙拦截：




工作原理:

标准的HTTP浏览模式如下：

1. TCP打开连接（三路握手信号）
2. HTTP发送GET命令
3. 数据下载到浏览器

在一个网页打开多个连接的情况下（大多数网页都是如此）。每个连接需要一个DNS 查询，响应，TCP SYN-SYN/ACK-ACK，以及HTTP GET。之后数据才会出现在显示屏上。

当你在packet detail面板没有看到显示内容时，右键报文并选择Follow TCP stream，会看到连接的细节数据。另一个广泛应用的工具是Fiddler，Fiddler是HTTP故障排查的免费工具。

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（二十九）：Wireshark抓包实例诊断数据库常见问题

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese



## 介绍

通常来说，数据库，应用和网络在IT架构中处于不同的分支。数据库的故障排查由DBA来完成，但是网络工程师仍可以从抓包中定位出问题并不出自网络。当IT抱怨“网速慢”，实际并不一定是这样。下文帮助我们验证所谓“网速慢”的问题。

## 更多信息

工作过程:

对于数据库问题，按照以下步骤：

1. 当怀疑是“慢速网络响应”时，问以下问题：
    - 问题发生于本地还是全局？是只发生在远端办公室，还是center也有发生？如果整个网络都出现问题，就不会是WAN带宽问题。
    - 对于所有客户端是否都发生了这样的问题？如果只是某些特定用户碰到问题，则可能是这些用户运行了某些应用导致。
    - 客户端和服务器之间通讯链路是否过载？导致过载的应用是什么？
    - 是所有应用都运行缓慢，还是使用特定数据库的应用运行缓慢？是PC比较老旧，还是服务器资源耗尽?
2. 搞清楚上述问题之后，开始故障排查：

    1. 打开Wireshark开始抓包。可以将对端端口连到PC，服务器，VLAN，或连接远端客户端的路由器。

    2. 在expert info中查看TCP事件。这些事件发生在整个通信链路，或是特定的IP地址，还是特定的TCP端口？此操作能够帮助定位问题并验证是否发生于特定链路，服务器，或是应用。测试连接到Internet的数据流时，可能会得到发往站点或mail server（诸如此类）的很多重传以及重复ACK。在组织内部，重传范围应当在百分之0.1至0.5。连接到Internet时，可能会高得多。

3. 当你看到网络上有问题时，按照前几张的故障排查步骤给予解决。但是，也有些网络问题会影响数据库操作。下例中，可看到客户端与服务器通信链路往返延时达到35至40ms。

1. 我们查看TCP stream 8（1），连接开始于TCP SYN/SYN-ACK/ACK。如下图（2）所示。可以看到整个连接花费了371个报文（3）。

Filter: tcp.stream eq 8 ① Expression... Clear Apply Save NBNS NBDS NBSS

| No. | Time | Source | Destination | Protocol | Info |
|---|---|---|---|---|---|
| 1840 | *REF* ② | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [SYN] Seq=0 W |
| 1844 | 0.013483 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [SYN, ACK] Se |
| 1845 | 0.013496 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [ACK] Seq=1 A |
| 1846 | 0.015710 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |
| 1847 | 0.044857 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [PSH, ACK] Se |
| 1848 | 0.045057 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |
| 1849 | 0.075752 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [PSH, ACK] Se |

Frame 1930: 87 bytes on wire (696 bits), 87 bytes captured (696 bits)
Ethernet II, Src: Hewlett-_3e:54:e7 (00:0b:cd:3e:54:e7), Dst: Cisco_4f:4a:ec (00:60:47:4f:4
Internet Protocol Version 4, Src: 192.168.20.88 (192.168.20.88), Dst: 192.168.10.80 (192.16
Transmission Control Protocol, Src Port: vfo (1056), Dst Port: wv-csp-udp-cir (3717), Seq:
Data (33 bytes)

File: "C:\Courses\Upstream Systems\PCAP Files\Example 0... | Packets: 6494 Displayed: 371 ③ | Marked: 0 Load time: 0:00.307 | Profile: Wireless

2. 连接继续，可见到DB请求与响应之间时间间隔大约35ms。

| No. | Time | Source | Destination | Protocol | Info |
|---|---|---|---|---|---|
| 1981 | 35.833309 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |
| 1982 | 35.869385 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [PSH, ACK] Se |
| 1983 | 35.869930 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |
| 1984 | 35.905654 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [PSH, ACK] Se |
| 1985 | 35.906194 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |
| 1986 | 35.944428 | 192.168.10.80 | 192.168.20.88 | TCP | wv-csp-udp-cir > vfo [PSH, ACK] Se |
| 1987 | 35.953804 | 192.168.20.88 | 192.168.10.80 | TCP | vfo > wv-csp-udp-cir [PSH, ACK] Se |

3. 由于往返已经有371个报文，371X35 ms大约是13秒。加上可能发生一些重传导致延时，用户查询一次数据库可能要等待10至15秒。

4. 这种情况下，应当与DBA讨论怎样大幅减少网络上传输的报文数量，或是改变终端服务器或网络接入的方式。

4. 另一个可能发生的问题是抓包文件反映出有软件问题。以下截图中可看到5个重传（1），并且客户端打开了一个新的连接（3）。看起来像一个TCP问题但只发生在软件中一个特定窗口。这只是由于一个软件进程停止运行，因此TCP无法对客户端作出响应（2）。

| No. | Time | Source | Destination | Protocol | Info |
|---|---|---|---|---|---|
| 274 | 0.078889 | 192.168.3.50 | 192.168.200.227 | TCP | http > vrtp [ACK] Seq=1 Ack=59884 Win |
| 275 | 0.380166 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP Retransmission] [TCP segment of |
| 276 | 0.983678 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP Retransmission] [TCP segment of |
| 277 ② | 2.195589 | 192.168.200.227 | 192.168.3.50 | TCP ① | [TCP Retransmission] [TCP segment of |
| 278 | 4.604757 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP Retransmission] [TCP segment of |
| 279 | 9.432867 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP Retransmission] [TCP segment of |
| 280 | 18.989050 | 192.168.200.227 | 192.168.3.50 | TCP | rcts > http [SYN] Seq=0 Win=65535 Len |
| 281 | 18.994054 | 192.168.3.50 | 192.168.200.227 | TCP ③ | http > rcts [SYN, ACK] Seq=0 Ack=1 Wi |
| 282 | 18.994085 | 192.168.200.227 | 192.168.3.50 | TCP | rcts > http [ACK] Seq=1 Ack=1 Win=655 |
| 283 | 18.994264 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP segment of a reassembled PDU] |
| 284 | 18.994280 | 192.168.200.227 | 192.168.3.50 | TCP | [TCP segment of a reassembled PDU] |

更多建议：

右键数据库客户端与服务器会话报文，会打开一个窗口，有助于DBA查看网络问题。当碰到延时问题时，例如，通过移动电话接入Internet，数据库客户端到服务器的通讯可能效率低下。可能需要切换接入方式。

很重要的一点是搞清楚数据库的工作模式。如果客户端正在接入数据库服务器，数据库服务器正在使用从另一台服务器共享的文件，可能客户端——服务器工作良好，但问题可能出在数据库服务器与文件服务器之间共享文件上。确保在开始测试之前确知所有依赖条件。

最重要的是，与DBA保持良好关系。

# 参考

Network Analysis Using Wireshark Cookbook

# 网络基本功（三十）：细说DNS（上）

转载请在文首保留原文出处：EMC中文支持论坛https://community.emc.com/go/chinese



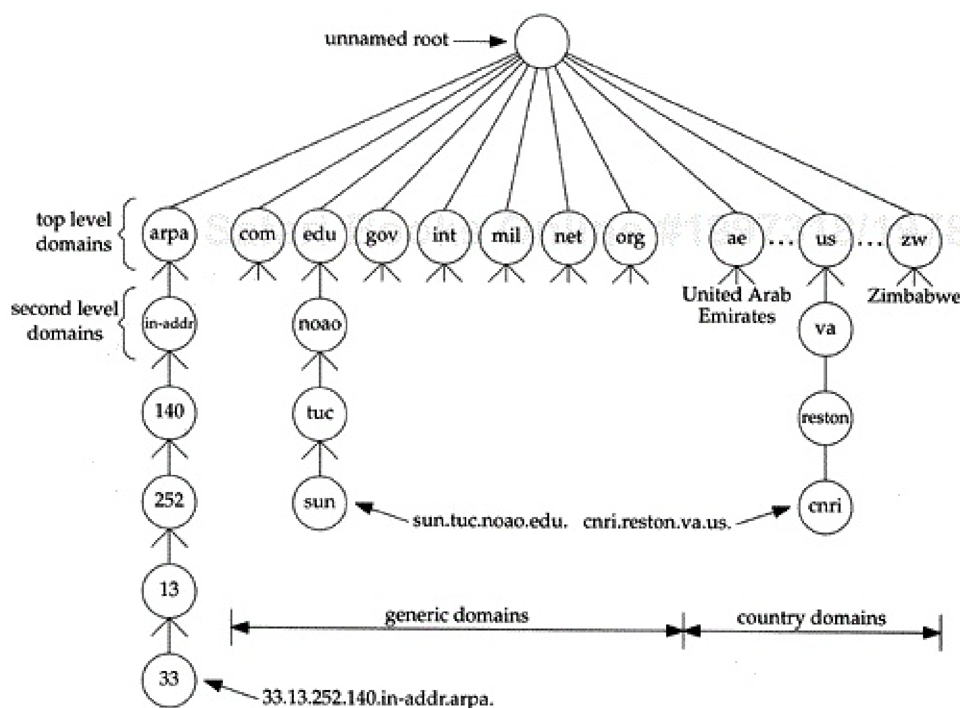

## 介绍

因特网上作为域名和IP地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的IP数串。通过主机名，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。

## 更多信息

**DNS基础:**

DNS命名空间是一个分层结构，类似于Unix文件系统。如下图的分层空间所示。




每个节点都有一个标签，最多可以有63个字符。树结构的根部是一个特殊的标签为null的节点。树结构中节点的域名就是一串标签列表，从该节点开始，一直到根节点，通过dot来将标签分开。（这是与Unix文件系统不同的地方，将路径名放在最前沿着树结构下来）。树结构中的每一个节点必须有一个唯一的域名，但是树结构中不同的point可以有相同的标签。

域名分为绝对域名与相对域名。绝对域名也称为完全合格的域名FQDN(Full Qualified Domain Name)，它是以“.”结尾的域名，例如sun.tuc.noao.edu.。如果不以“.”结尾，则假设该域名需要被补充完整。域名如何补充则取决于使用的DNS软件。

顶级域名分为三个区域：

1. arpa是用来做反向域名解析的特殊域。
2. 七个3字母域名称为普通域名，也有称为组织域。
3. 所有两个字母域名是基于ISO 3166国家代码，称为国家域名或地理域名。

上图中没有显示的很重要的一点是DNS中责任的分派。没有一个单一的实体来管理树中的每一个标签。相反，一个实体（网卡）维持树中的一部分（顶级域名）并将其他责任分配给zone中其他实体。

zone指DNS树中分开管理的子树。例如，二级域名就是一个常见的zone，noao.edu。很多二级域名又分为更小的zone。例如，一所大学按照系别，公司按照部门分为分为更小的zone。

熟悉Unix文件系统的会注意到DNS树按zone分区很像逻辑Unix文件系统分为物理磁盘分区。如同我们从上图中无法看出zone的委托授权管理位于何处，从Unix文件系统的类似图中也难以看出哪个目录在哪个磁盘分区上。

一旦zone的委托授权分派好，zone的负责人需要为其提供多个域名服务器。当zone中安装了新的机器，zone的DNS管理员为其分配域名与IP地址，并将信息输入域名服务器的数据库中。域名服务器委托授权管理一个或多个zone。zone管理人员必须为其提供一台主域名服务器以及一个或多个二级域名服务器。主服务器和二级服务器必须相互独立并冗余，以使zone不会受到单点故障的影响。主服务器和二级服务器的区别在于，主服务器从磁盘文件加载zone的所有信息，而二级服务器从主服务器获取所有信息。这一过程称为zone transfer。

当新的机器添加到zone中，管理员将合适的信息（至少需要名称和IP地址）添加到主服务器系统的磁盘文件中。之后告知主域名服务器重新读取自己的配置文件。二级服务器定期查询（通常3小时一次），如果主服务器有新的数据，二级服务器通过zone transfer来获取。

当域名服务器没有所需信息时怎么办呢？它必须联系另外一台域名服务器。这是DNS的分布式特性。并不是每一台服务器都知道如何联系其他域名服务器，但每一台服务器都知道如何联系根域名服务器。根服务器的IP地址存放于主服务器的配置文件中。主服务器必须知道根服务器的IP地址，而非DNS名。之后，主服务器获知所有二级域名服务器的名称和位置（即IP地址）。整个交互过程是：发起请求的域名服务器必须联系根服务器，根服务器告知请求服务器联系另外一台服务器，这样逐级进行。

DNS的一个基本属性是缓存。即，当一个域名服务器收到一条映射信息（如一个主机名的IP地址），它会将该信息放入缓存，以使之后的查询可以使用缓存后的结果，而无需额外发起对其他服务器的查询。

# 网络基本功（三十一）：细说DHCP

转载请在文首保留原文出处：**EMC**中文支持论坛https://community.emc.com/go/chinese




## 介绍

动态主机设置协议（Dynamic Host Configuration Protocol, DHCP）是一个局域网的网络协议，使用UDP协议工作，主要有两个用途：

- 给内部网络或网络服务供应商自动分配IP地址给用户
- 给内部网络管理员作为对所有电脑作中央管理的手段

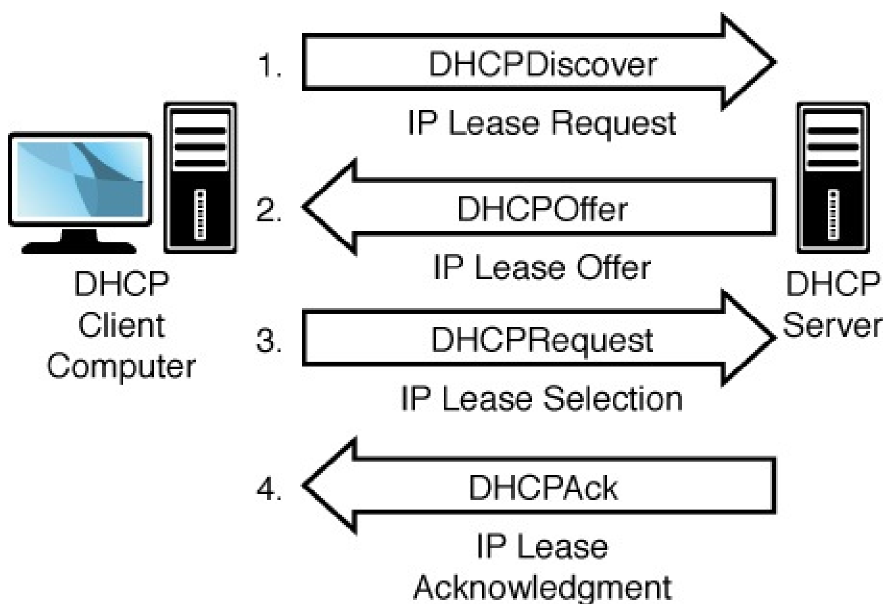
本文介绍DHCP的工作原理。

## 更多信息

**DHCP**工作原理**:**

DHCP从一个IP地址池中提供IP地址，该池有DHCP服务器数据库定义，称为scope。如果客户端接受这一地址，则它可在一个预定义的期限内使用该地址，称为租约。如果客户端无法从DHCP服务器获取IP地址，它就无法正常初始化TCP/IP。

在DHCP为客户端配置TCP/IP参数时，DHCP服务器和客户端都需要经历四步过程。注意到很多通讯是通过广播的方式来完成的。如果路由器无法转发这些DHCP消息时，广播通信可能会造成问题。



当客户端处于以下四种状态之一时，必须使用IP租约进程：

- 配置使用DHCP的客户端第一次初始化TCP/IP；
- 客户端请求特定的IP地址但服务器拒绝了该地址，在DHCP丢弃租约时即会发生。
- 客户端之前租约了一个IP地址，但之后释放了该IP地址，现申请一个新的租约。这种情况发生于用户输入ipconfig /release和ipconfig /renew命令时。

客户机请求**IP**地址（**DHCPDISCOVER**）：

当一个IPv4客户机启动时监测到需要IP地址，它会初始化一个TCP/IP的限制版本，之后广播一个报文请求寻找DHCP服务器的地址。该广播报文告知监听服务器客户端需要IP地址信息。DHCP客户端发送的报文这一阶段包括租约请求，客户端源地址，0.0.0.0，目的地址，即广播地址255.255.255.255。报文也包括客户端硬件MAC地址和机器名，该信息也指明了向DHCP服务器发起请求的设备。

客户端向DHCP服务器发送请求IP地址的真实报文称为DHCPDISCOVER报文。网络上每一台安装了TCP/IP协议的主机都会接收到这种广播信息，但只有DHCP服务器才会做出响应。

服务器提供**IP**地址（**DHCPOFFER**）：

所有拥有有效IP地址的DHCP服务器都会向DHCP客户端提供IP地址信息。它响应以地址池中一个未分配的IP地址供请求主机使用。要能够响应DHCPDISCOVER报文，DHCP服务器必须拥有客户端的有效IP配置信息。DHCP服务器回复的DHCPOFFER报文包含以下信息：

- 客户端的硬件地址
- 提供的IP地址
- 合适的子网掩码
- 租约有效期
- 服务器ID，即DHCP服务器的IP地址

客户机选择**IP**地址（**DHCPREQUEST**）：

DHCP客户端选择它所接收到的第一个DHCPOFFER报文提供的IP地址。之后，它把这一信息广播至网络。该报文中，客户端请求服务器提供给它的IP地址。这是因为客户端可能收到不止一个DHCP服务器发送的offer。通过广播这一请求，客户端告知其他DHCP服务器不会再接受其他offer。为了进一步确保客户端接受的服务器offer没有疑义，DHCPREQUEST报文中还包含以下信息：

- 提供所接受offer的服务器IP地址
- 客户端硬件地址
- 客户端接受的IP地址

服务器确认**IP**租约（**DHCPACK**）：

DHCP服务器对客户端作出响应，将IP地址分配给客户端。之后，它发送DHCPACK确认信息给客户端。该信息包含IP地址的有效租约以及其他配置信息。

有时，在客户端接收服务器提供的租约后，DHCP租约请求仍可能不成功。可能有以下几种情况：

- 由于客户端移动至其他子网，IP地址无效
- 客户端尝试租约它之前的IP地址但该IP地址不再可用

在上述情况下，服务器会发送一条不成功信息DHCPNACK。收到DHCPNACK的客户端必须重新开始整个DHCP初始化进程。也就是说，它必须发送另一个DHCPDISCOVER报文查找新的IP地址。